SDI

SYSTEM DYNAMICS, INT'L., INC.

Inertially Aided Robotics

FINAL REPORT

by

Don N. Pittman and Dr. James C. Hung

SYSTEM DYNAMICS INTERNATIONAL, INC.

Report Number

H-90-1

Prepared for

U.S. ARMY MISSILE COMMAND
Redstone Arsenal
Huntsville, AL

under

Contract No. DAAH01-88-D-0057
Task No. 0037

December 31, 1989

Inertially Aided Robotics

FINAL REPORT

for

Contract No. DAAH01-88-D-0057
Task No. 0037

DISTRIBUTION LIST

U.S. Army MICOM

Guidance & Control Directorate	4
Redstone Scientific Information Center (RSIC)	1

System Dynamics International, Inc.

Huntsville, AL	1
Gainsville, FL	1

Table of Contents

List of Figures

List of Tables

Executive Summary

Currently, the control of the position and orientation of the end-effector of a robot manipulator is usually done by controlling the joint angles of the manipulator. This approach suffers from load disturbances, changes in manipulator parameters, and the effect of compliance. The approach is an open-loop control as far as the end-effector is concerned. Closed-loop control of the end-effector can be achieved by using an inertial measuring system with the sensors situated at or near the end-effector, thus improving the overall system accuracy and robustness.

This report documents the results of a task to provide engineering support for the continued development of a prototype inertially aided robotic end effector position determination system. This support included the design and fabrication of a special data acquisition system (DAS), analyzing the effects of system parameters on position accuracy, developing and implementing real-time position determination software, and integrating hardware and software into a single-axis robotic end-effector position determination system.

The DAS was designed to provide high resolution (18 bits effective, 22 bits internal), high accuracy, low drift analog-to-digital conversions at up to 500 samples/sec. Software was also developed to allow this hardware to be used for data collection.

A study of position accuracy on system parameters such as accelerometer scale factor (SF) and bias stability, SF nonlinearity, filter bandwidth, sampling rate, and resolution was also conducted. This study was useful in defining the parameter limits that are needed for different applications.

A single-axis position determination system which consisted of 1) a Q-flex QA2000 accelerometer, 2) the DAS, 3) a single-axis linear translation table with an position encoder and 4) real-time position determination software. The real-time software was a menu driven program that included zero velocity updates, temperature compensation, experiments of five different application scenarios, and the capability of displaying, transmitting, or storing the position updates in real-time.

The test results of the system were below that expected of the DAS. Therefore, the dominant error sources seem to reside in the accelerometer. An accelerometer that is specifically designed for this application (operation range of ± 2 Gs) is needed. Also, from the simulation of the DAS, it was determined that 20 bits resolution is more than needed and a 100 Hz sampling rate is too slow for real-time accuracy. Hence, using a 16-bit converter to sample at 20 kHz and then averaging 4 samples resulting in a 5 kHz acceleration update should be more accurate than the current system for real-time position accuracies less than 10 thousands of an inch. The averaging scheme will also tend to increase the effective resolution of the system. Before continuing any more hardware development an in-depth simulation of the entire system (open and closed loop) from the accelerometer to the processor should be conducted. This approach would better define the propagation of error sources to the position error.

1 Introduction

For most of the current robot manipulators, control of its end-effector position and orientation is done by controlling joint angles as shown in Figure 1. Each joint is controlled by a local joint servo. Angular position sensors are installed at manipulator joints to measure joint angles. For a desired end-effector position and orientation, inverse kinematics is used to generate command signals in joint coordinates [1,2]. These signals become the reference inputs to local joint servos. Such control scheme may be called "joint sensor based manipulator control". Figure 2 shows a block diagram of this control scheme. Note ~~that although~~ local feedback exists in each joint servo, there is no feedback to compare the actual end-effector state with respect to the reference state.

End-effector control without feedback suffers from two major shortcomings. The first is the effect of arm compliance on the control, and the second is poor robustness. The compliance is caused by the physical nonrigidness of the manipulator and by the insufficient stiffness of the joint servos. The compliance effect causes two problems. The first is the bending and/or drooping of manipulator arms caused by loading and by the weight of arms. This affects the accuracy of positioning the end-effector. The second problem is the existence of bending modes in the manipulator's dynamical characteristics, making an accurate and steady control of the end-effector difficult. Current methods to cope with the problems of compliance is to adopt large size arm cross-sections, resulting in a bulky manipulator.

The lack of robustness is a well known nature of any system using open-loop control. The system is incapable of coping with the load disturbance and the changes in plant parameters. Both can be severe in a manipulator system.

Closing the end-effect loop can be done by optical means. An optical position monitor consists of one or more cameras and an image processing microcomputer. A three-dimensional picture of the end-effector is taken by cameras, converted into digital data, and processed by microcomputer to generate the command signal for the manipulator. This arrangement has its drawbacks. First of all, it requires ample computation effort, thus reducing the bandwidth of the measured data. As a result, the data may not be useful for bending mode control. Secondly, there are situations where uses of cameras are not feasible. Therefore, closing the end-effect loop by cameras is not always an effective approach for the improvement of manipulator robustness.

This report documents the results of a task to provide engineering support for the continued development of a prototype inertially aided robotic end effector position determination system. This support included the design and fabrication of a special data acquisition system (DAS), analyzing the effects of system parameters on position accuracy, developing and implementing real-time position determination software, and integrating hardware and software into a single-axis robotic end-effector position determination system. The concept has numerous advantages as compared to the joint sensor based control. However, implementation of the concept requires the solution of some practical problems. The problems and the attempted solutions for them will be discussed. Test results will also be presented.

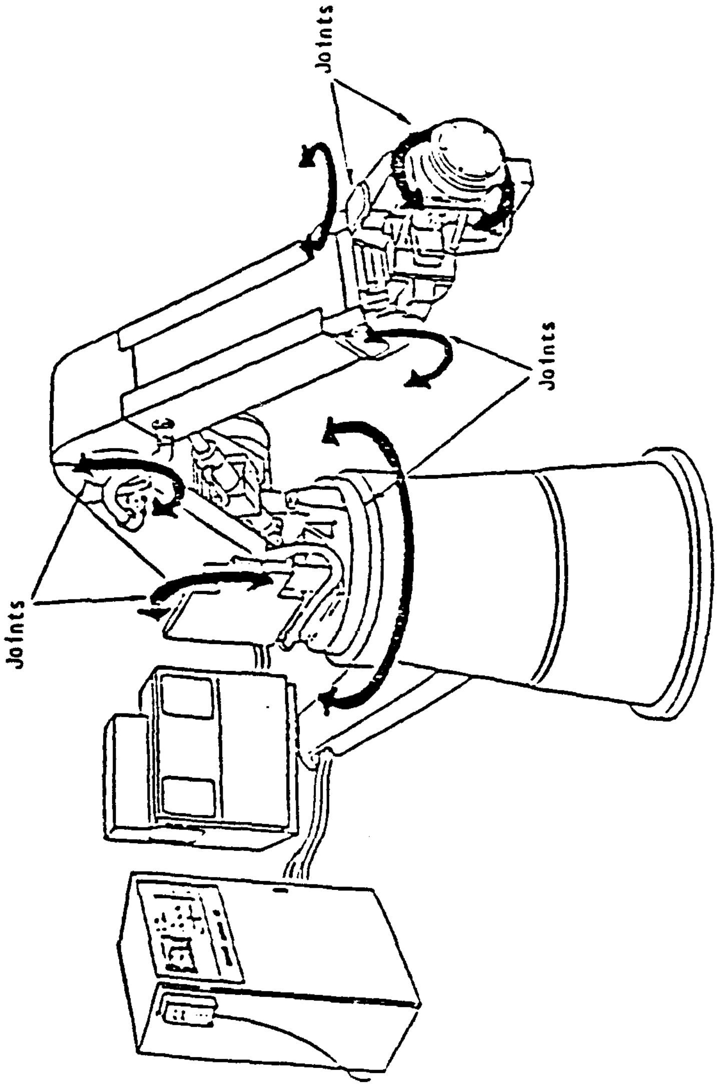


Figure 1 - Robot Manipulator having Joint Sensor Based Control

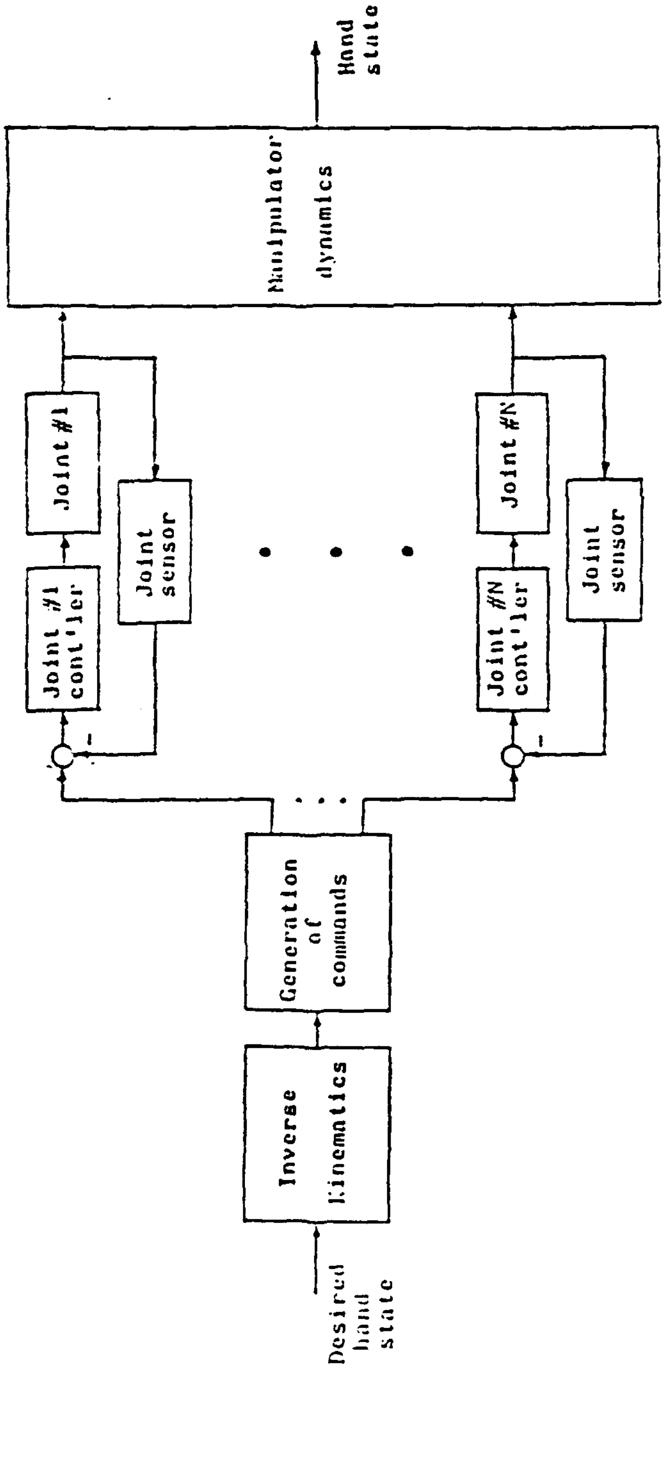


Figure 2 - Block Diagram of Joint Sensor Based Manipulator Control

2 Advantages of Inertially Aided Robotics

A concept has been proposed to close the end-effector feedback loop using an inertial measurement system (IMS) [3]. An IMS consists of two parts, an inertial measurement unit (IMU) and a data processing microcomputer. The IMU is the sensor which measures both the linear and rotational motions of its case while the microcomputer processes the IMU output data for the state (including position and attitude) of its case. By mounting an IMU at or near the end-effector of a manipulator, as shown in Figure 3, the state of the end-effector can be determined. By comparing the actual end-effector state and the desired state, errors are generated, which are processed by microcomputer to generate the commands for joint sensors. In this approach, precision requirement for joint sensors, needed for joint servos, can be greatly reduced since the error of end-effector state is sensed by the IMU and can be made independent of the errors of joint sensors. This approach may be called the "inertial measurement (IM) based manipulator control". Figure 4 depicts a block diagram of this approach. Note that total system feedback exists in the arrangement, which is markedly different from the local feedback in joint sensor based control. It is clear that IM based control is capable of coping the effects of arm compliance and capable of providing the desired robustness in control. The advantage of IM based manipulator control offers many practical features not available from the joint sensor based control. Some of them are given below:

1. The potential of handling all problems caused by arm and joint compliances. This includes improving robustness of the system with respect to manipulator loading, supporting the control of bending modes, simpler implementation of learning and repeating procedures, and stiffer end-effector control.

2. Providing signals for the stabilization of the end-effector of a manipulator on a moving platform (Figure 5). In fact, it can support the overall navigation of a mobile robot.

3. Relaxing the need for a complex analytic model of the manipulator and enabling the use of a simpler algorithm for precision end-effector control.

4. Relaxing the precision requirement of joint sensors.

5. Allowing the use of lighter arms, thus reducing the bulk and weight of the manipulator.

6. The potential of implementing a long stick end-effector for reaching a distant point (Figure 6).

It is clear that a successful development of an IM based control will have a significant impact on robot technology.

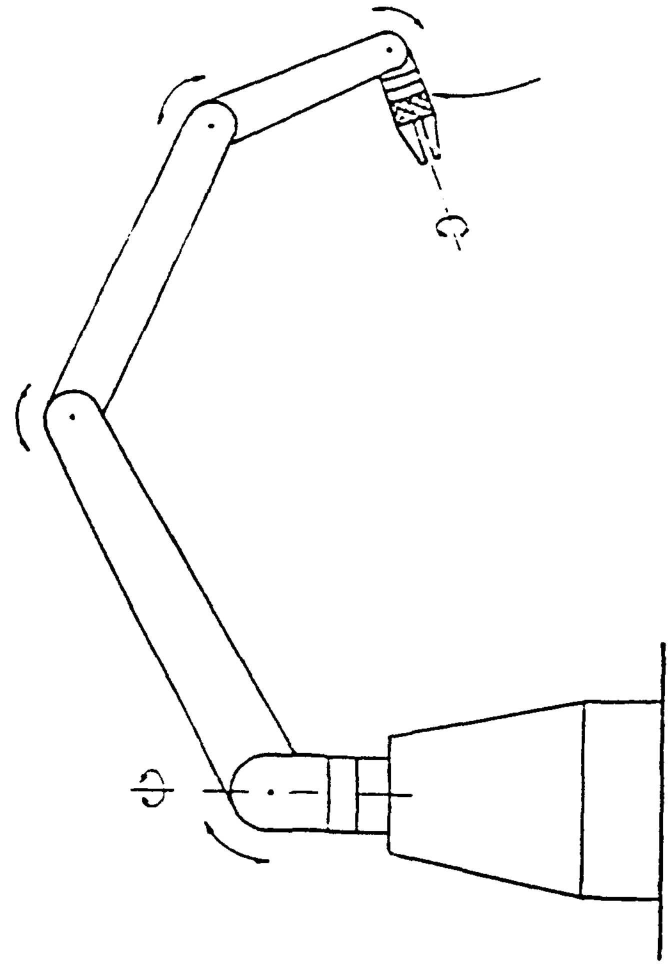


Figure 3 - Robot Manipulator Having Inertial Measurement Based Control

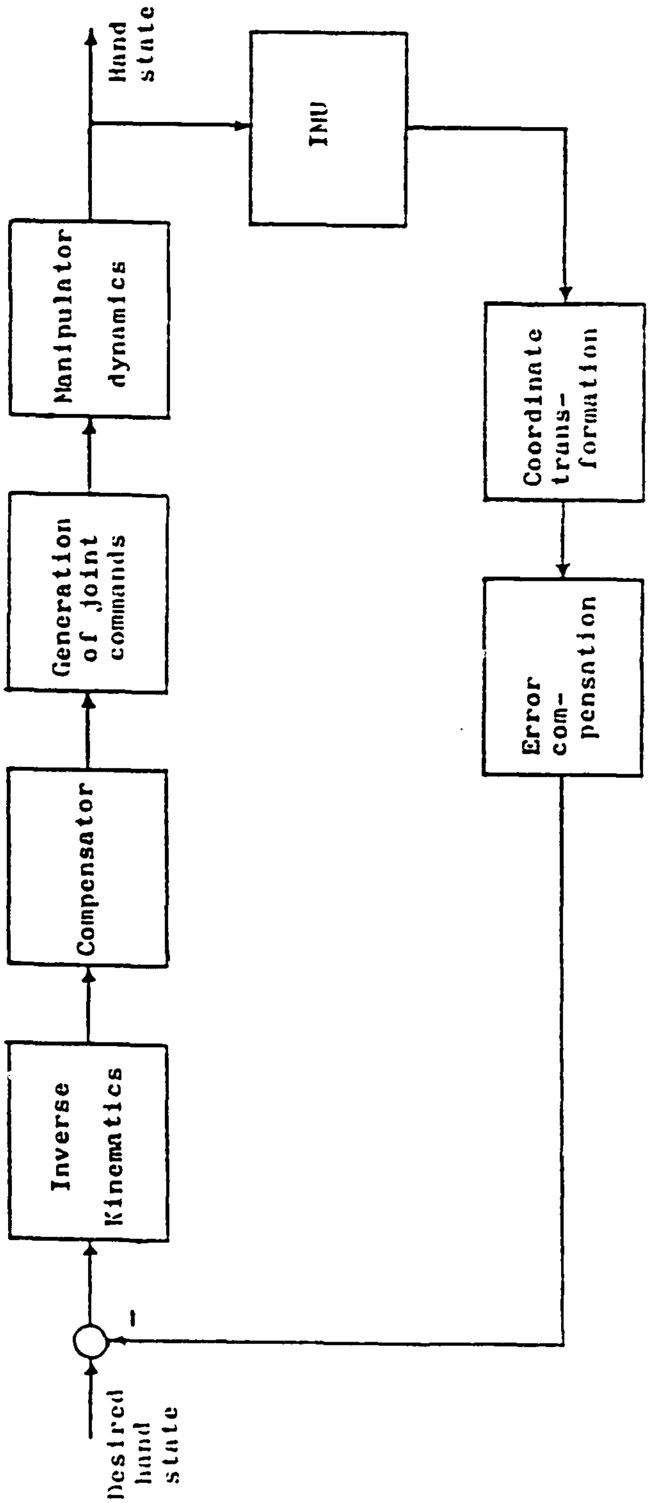


Figure 4 - Block Diagram of Inertial Measurement Based Manipulator Control

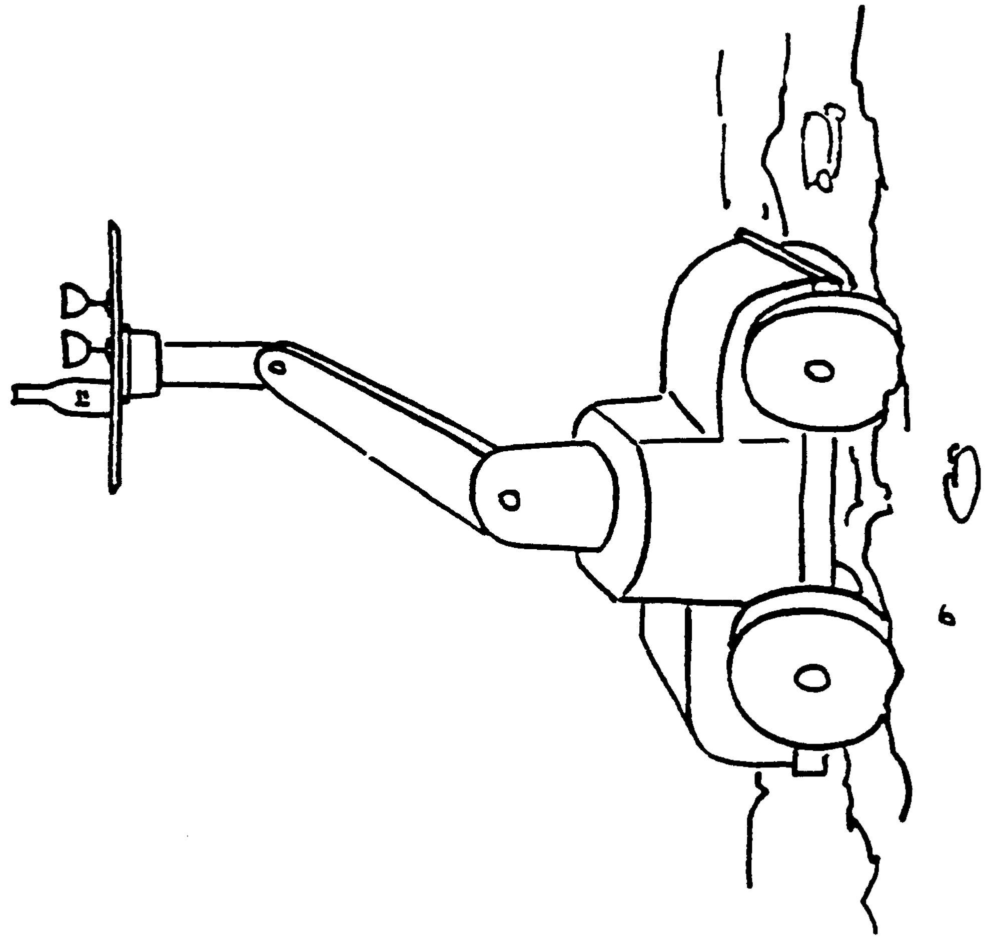


Figure 5 - IM Based Stabilization of End-effector on a Moving Platform

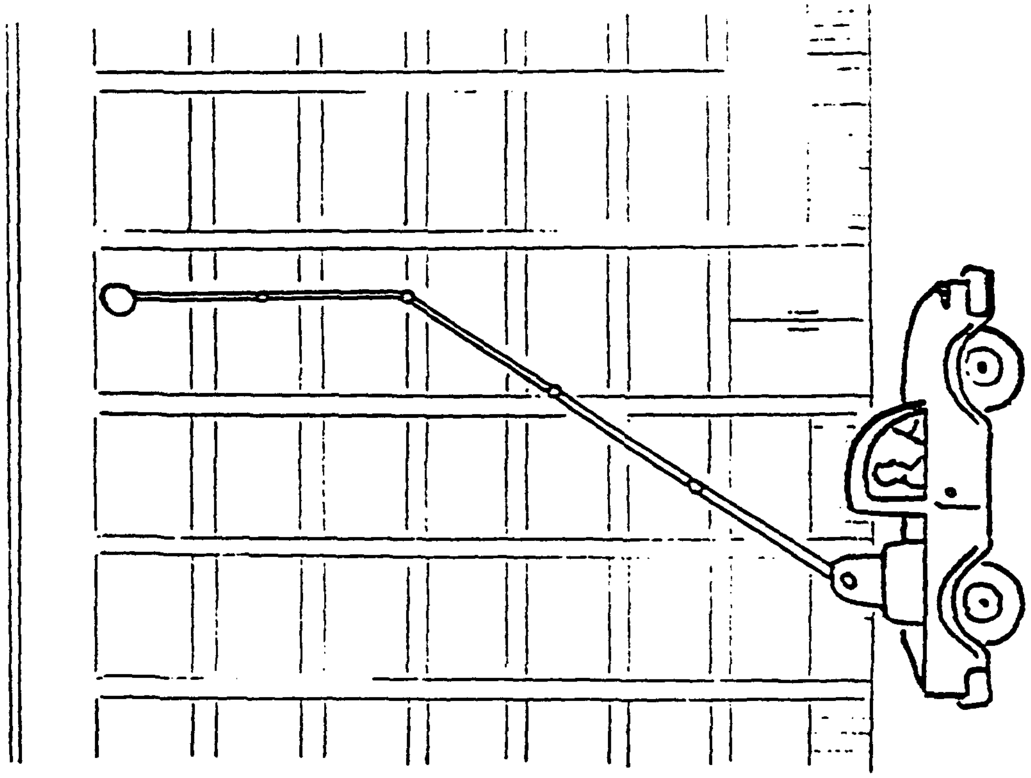


Figure 6 - IM Based Control of a Long Stick End-effector

3 Accelerometer Characteristics

3.1 Principle of Operation

A functional block diagram of the QA2000 accelerometer from Sunstrand Corporation is shown in Figure 8. The flexure and proof mass are made by etched amorphous quartz. An acceleration applied parallel to the sensitive axis will cause the proof mass to bend about the flexure. This movement unbalances the inputs into the position detector due to the change in the capacitance. The output of the position detector then drives the torquer amplifier until the proof mass has returned to the null position. The output of the torquer amplifier is proportional to the input acceleration and is therefore use as the output signal. A simple resistive load can be used to convert this signal into a voltage. The QA2000 package outline and pinout are shown in Figure 9.

3.2 QA2000 Performance Parameters

The Q-flex QA2000 accelerometer from Sunstrand is capable of measuring accelerations up to 1 kHz with minimal magnitude and phase errors. This fact can be noted from the frequency response plot in Figure 7. As will be discussed in the section on **system parameters** it is important to be able to measure the signal power from dc to > 1 kHz without degrading the magnitude and phase characteristics. Therefore future designs should incorporate even higher frequency devices than the QA2000.

One of the most important properties of an accelerometer for this type of application is low noise. Accelerometer noise is the dynamically changing output of the accelerometer that is not related to the actual input acceleration. This noise can be measured using a frequency analyzer such as the HP3562A from Hewlett Packard which was used for this task. The power spectrum (PS) of the accelerometer output without any filters was measured for a nominal dc input acceleration of 0.0. The PS for two different frequency ranges is shown in Figures 10 - 11. The two largest noise components at 27 Hz and 120 Hz are equivalent to sinusoidal accelerations with peak magnitudes of 98 μG and 118 μG. Integrated twice these noise components result in sinusoidal position errors with peak magnitudes of less than one thousandth of an inch. The reason these "large" acceleration errors have a negligible effect on the position error is the double integration effectively divides the magnitude of the noise components by the square of the frequency. Therefore, the problem occurs for noise components less than 1 Hz which are determined by the short-term bias and SF stabilities of the accelerometer. This problem is compounded by the fact that any noise components near multiples of the sampling frequency get "folded" down near dc. (This phenomenon is known as aliasing). Hence, filters should be used to filter out the frequencies above half the sampling rate. The total noise power, which would be the combination of all frequencies, can be determined by integrating the power spectrum from 0 to infinity. Taking the square root of the total noise power will yield the rms value. The rms value was approximated by measuring the peak to peak value of the output and then dividing by 6 resulting in 1.2 mG rms. This method of approximating the rms value assumes the amplitude probability density function to be Gaussian.

Table 1 presents the typical performance parameters of an accelerometer for the QA2000.

Table 1 - QA2000 Typical Performance Parameters

Parameter	Value
Scale Factor (SF)	1.25 mA/G
SF Temp Coef	120 PPM/°C
SF Stability	500 PPM
Bias Stability	500 µG
Bias Temp Coef	30 µG/°C
Range	± 25 G
Misalignment	2 mrad
Temp Range	-55 to 95°C

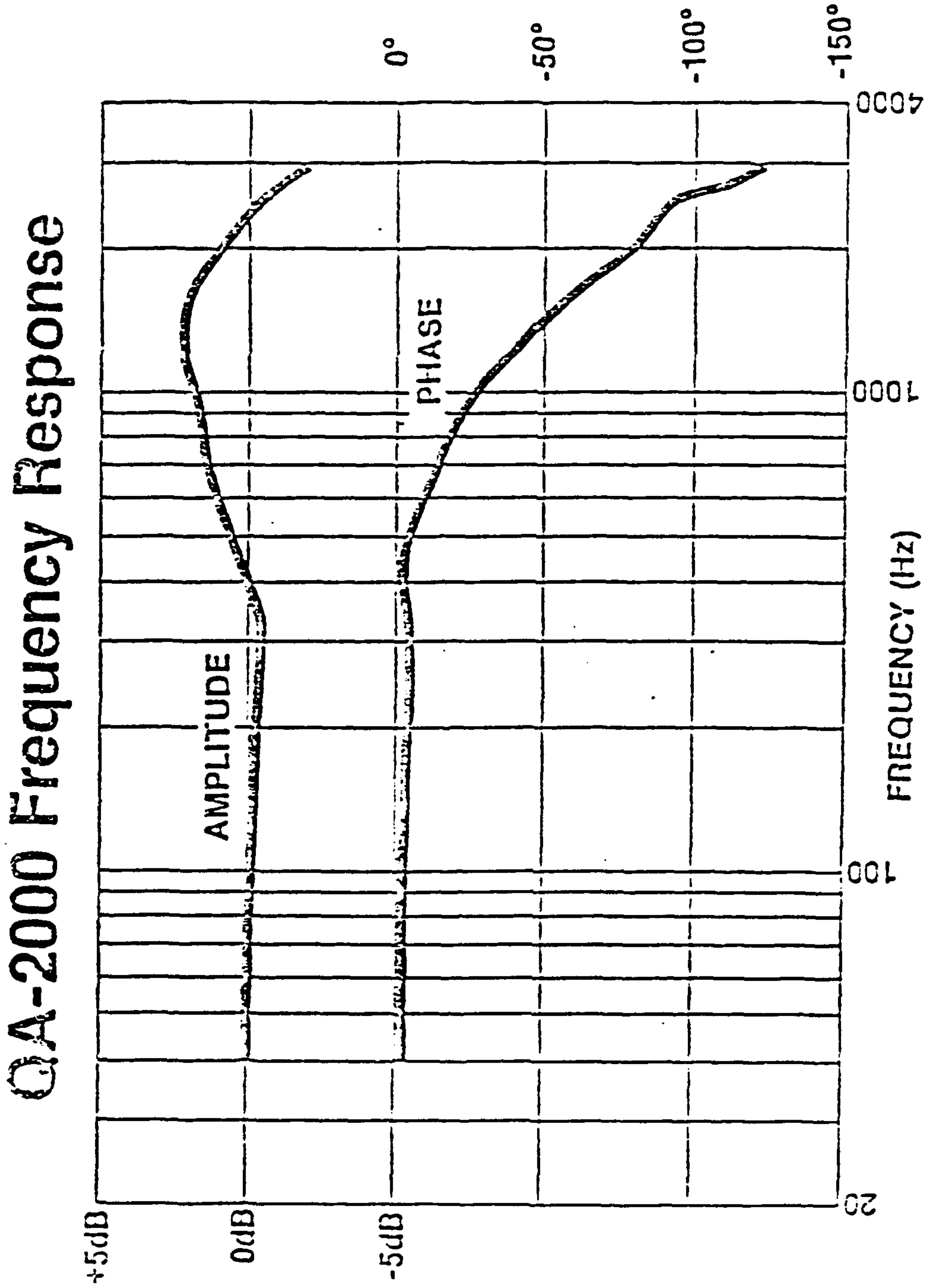


Figure 7 - QA2000 Frequency Response

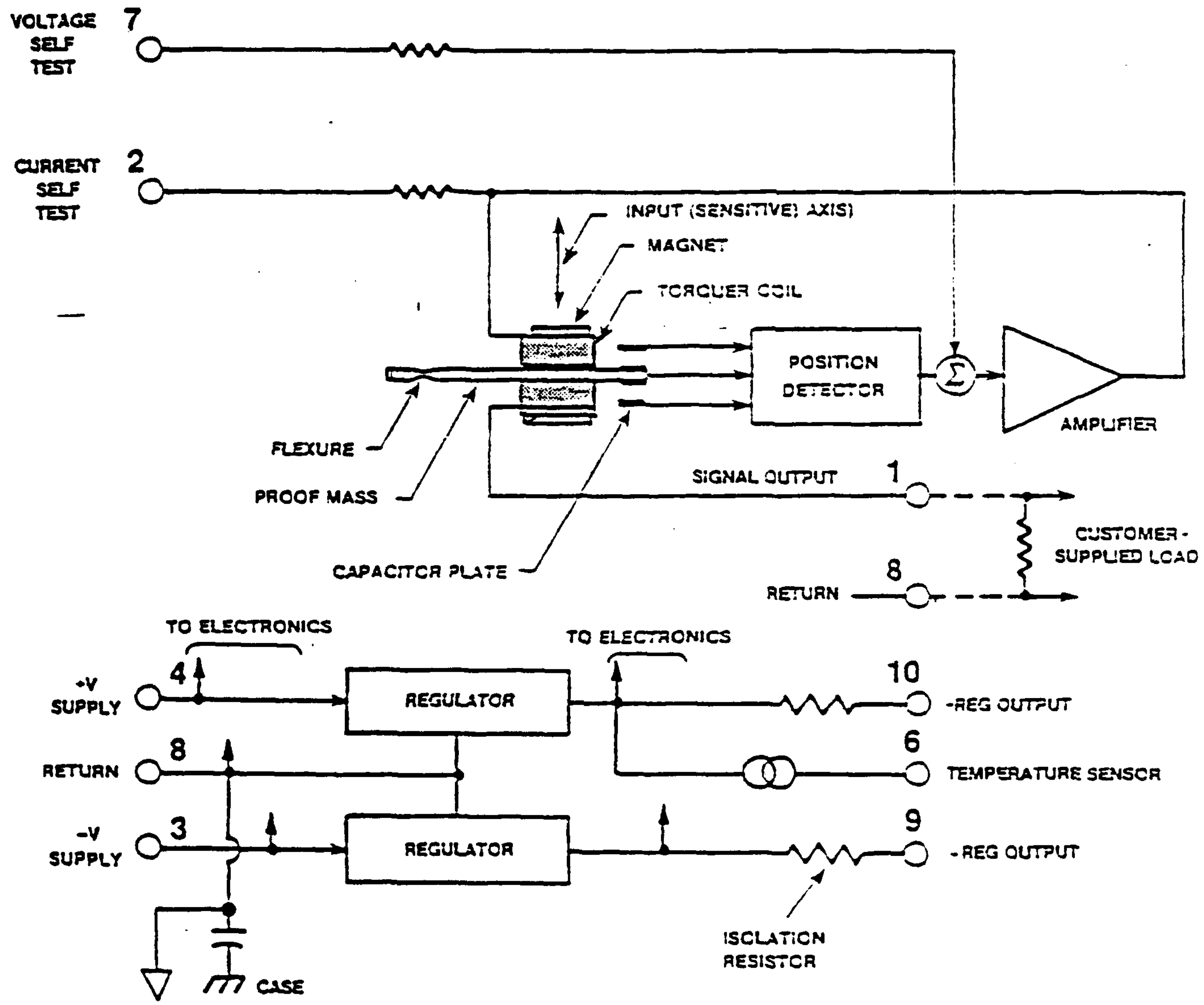
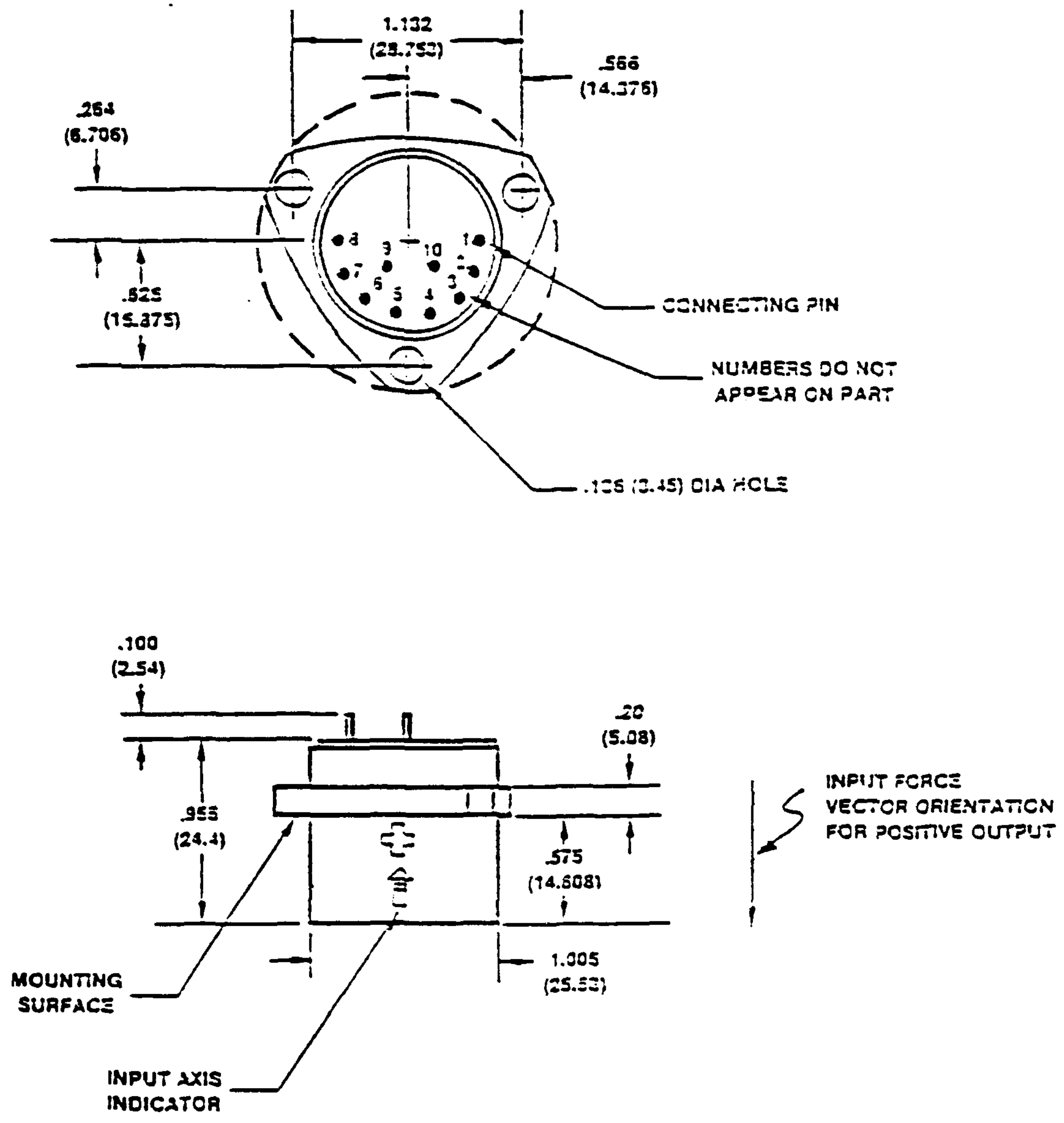


Figure 8 - QA2000 Functional Block Diagram



VIEW D (QA 1400, QA 2000)

Figure 9 - QA2000 Package Outline and Pinout

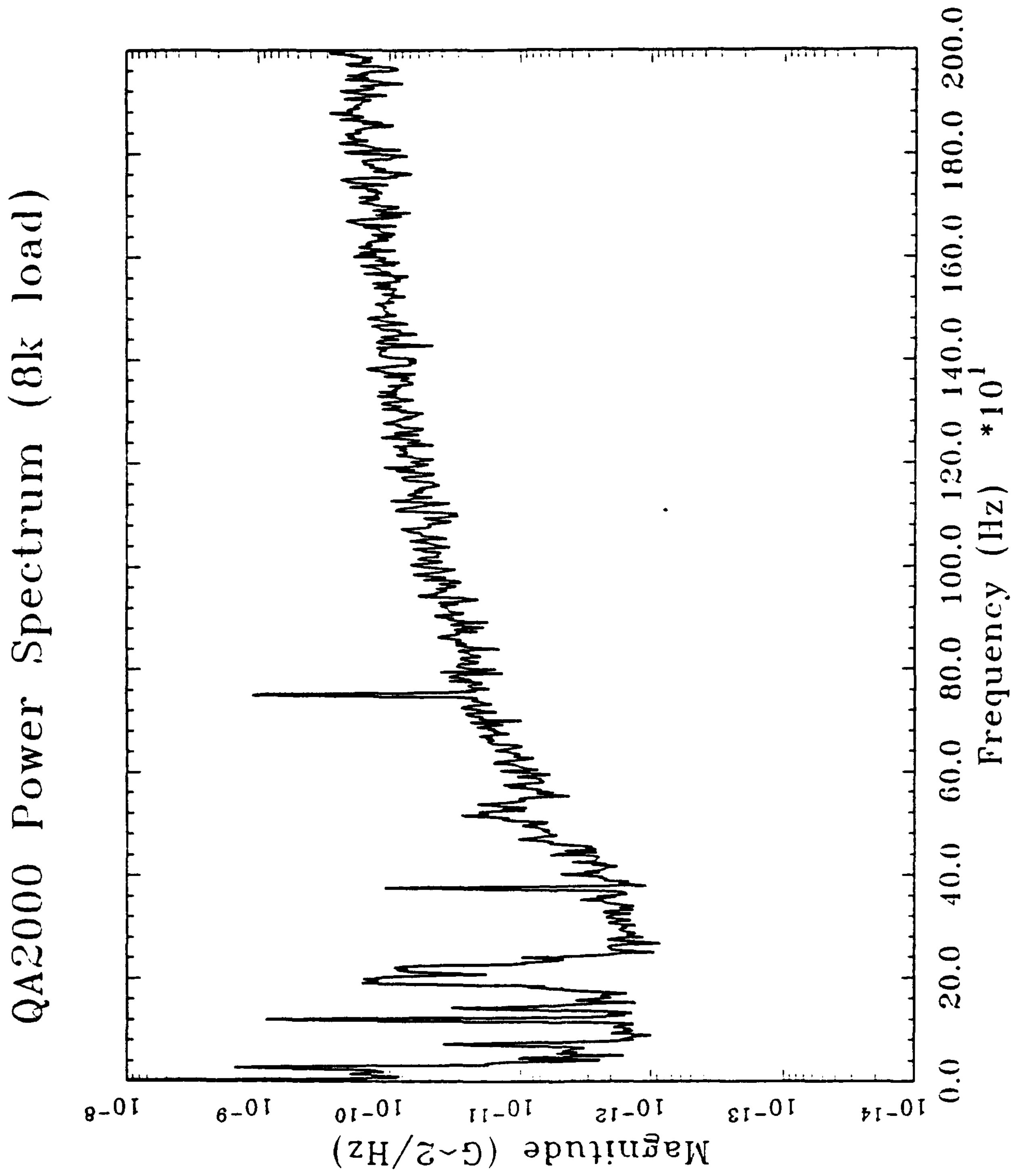


Figure 10 - QA2000 Power Spectrum (0 to 2 kHz)

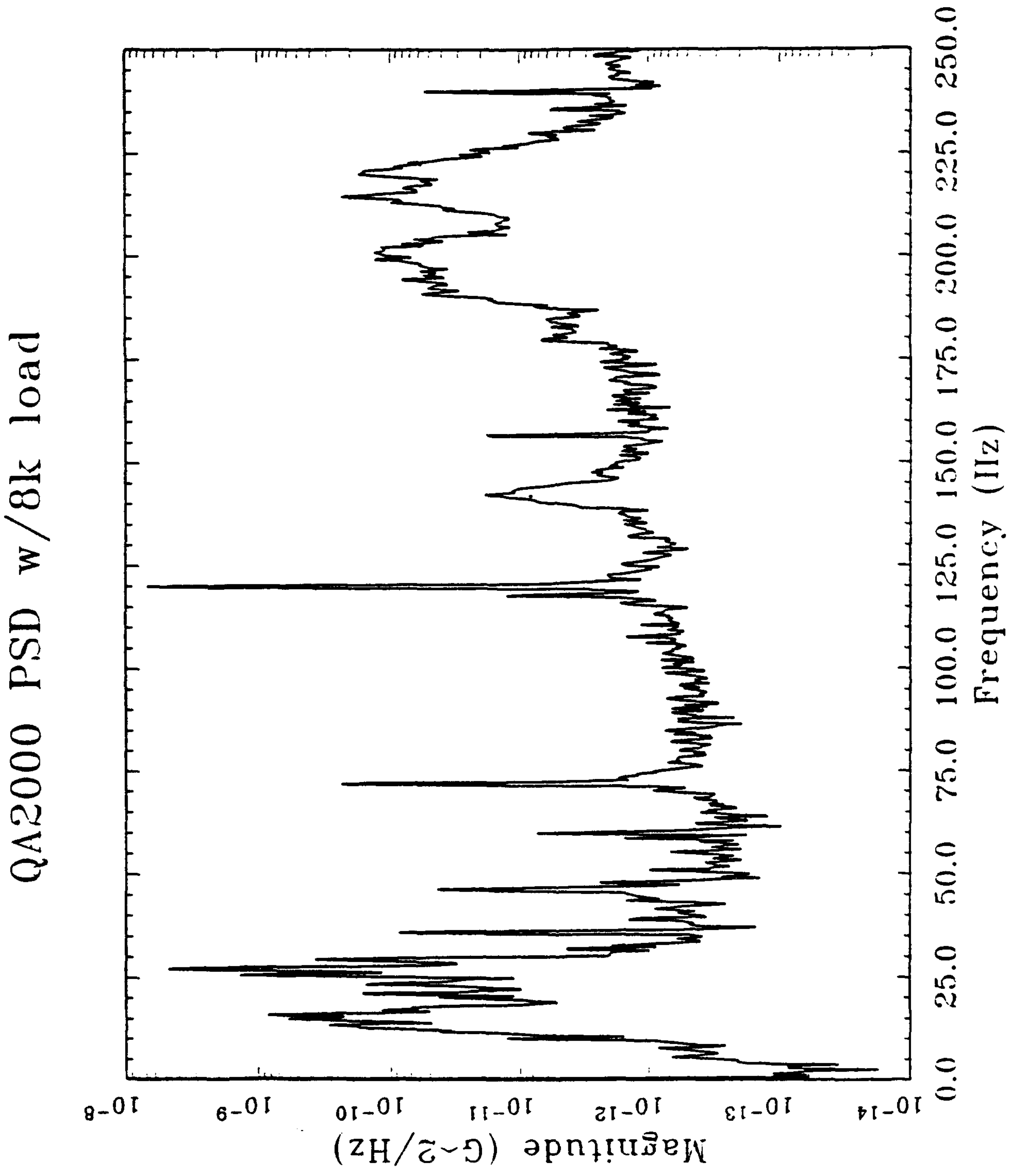


Figure 11 - QA2000 Power Spectrum (0 to 250 Hz)

4 Data Acquisition System

4.1 System Overview

A graphical overview of the data acquisition system (DAS) is provided in Figure 12 with the supporting schematics presented in Figures 13 - 19. A wiring list of the complete system can be provided on request. The system utilizes two AD1170 analog-to-digital convertors from Analog Devices to provide high resolution (up to 18 bits) sampling at rates up to 500 Hz. The A-to-D converters are triggered 180^o degrees apart so that the effective sampling rate is twice that of one convertor.

The system consists of four main sections, 1) the preamplifier and filters, 2) the AD1170 interface, 3) the 16-bit programmable counter/timer, and 4) the address decoder. The preamp and filters were specifically designed for used with the QA2000 accelerometer. The AD1170 interface allows for programming the converters. The counter/timer is used to set up a precise conversion timing signal and the address decoder provides the interface to an IBM PC/XT/AT or compatible. The following four sections present a detailed discussion of these circuits while the last two sections provide the component layouts and a list of system software.

4.2 QA2000 Preamplifier and Filters

The analog signal processing circuits are shown in Figures 13 and 14. In the schematic drawings the symbol used for analog ground is a triangle while the digital ground symbol was parallel lines in the shape of a triangle. The current-to-voltage preamplifier (U26) has two jumper selectable gains (JMP4) of 4000 and 8000 volts/amp each with a single pole cutoff at 3 kHz.

The jumper selections JMP1-3 provide a dc offset for the preamp that will cancel the dc signal measured by the accelerometer if the sensitive axis of is parallel with gravity. This cancellation allows the QA2000 to sense gravity without having to increase the range of the A-to-D which would decrease the resolution of the system. A better gravity compensation scheme is shown in Figure 20. This scheme involves averaging the accelerometer output (during zero velocity update) and then outputting the negative of this average to a digital-to-analog convertor to bias the preamp and cancel the gravity component. This method allows for any component of gravity to be cancelled during normal robot operation.

Another amplifier (U31) with a gain of 5 volts/volt was placed after the filters to allow for finer range and resolutions. A toggle switch (SW2) on the DAS box allows either the QA2000 current output to be measured or an external voltage signal. This external input is not connected to the preamp or filters, but it is connected to the x5 amplifier. Table 2 shows the proper jumper setting for each possible G range and Table 3 provides the jumper selections for gravity compensation.

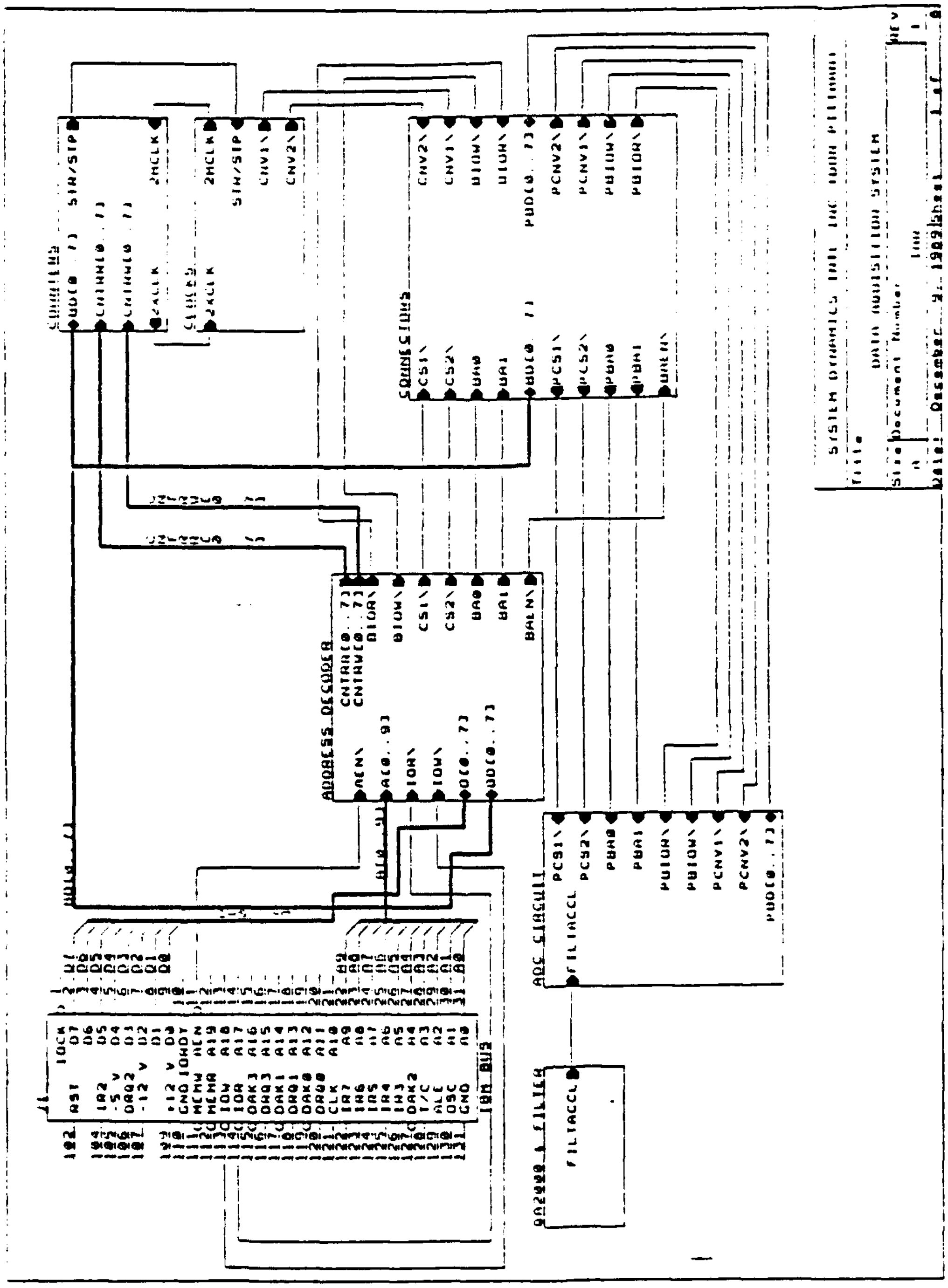


Figure 12 - Data Acquisition System Overview

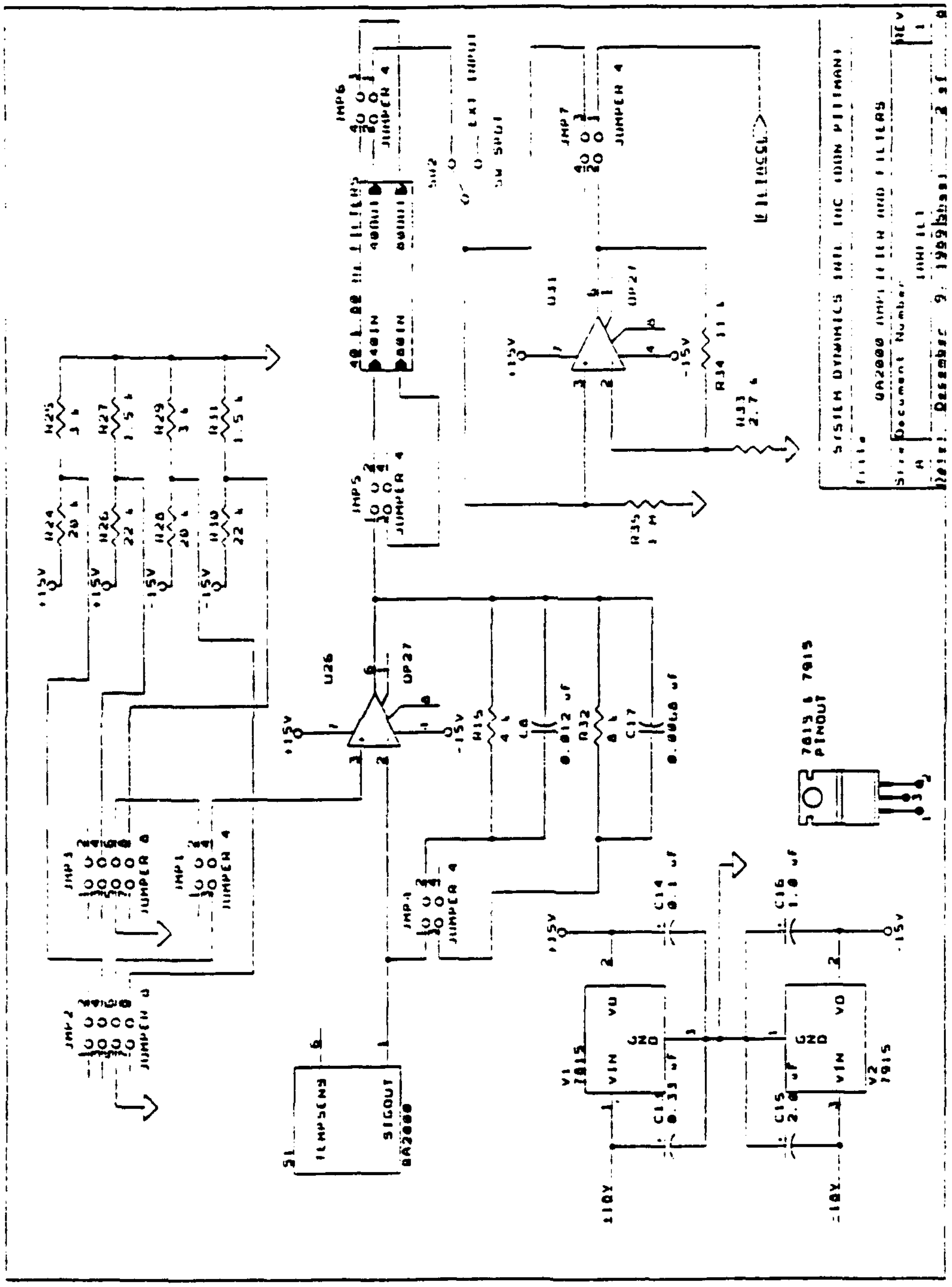


Figure 13 - QA2000 Preamplifier

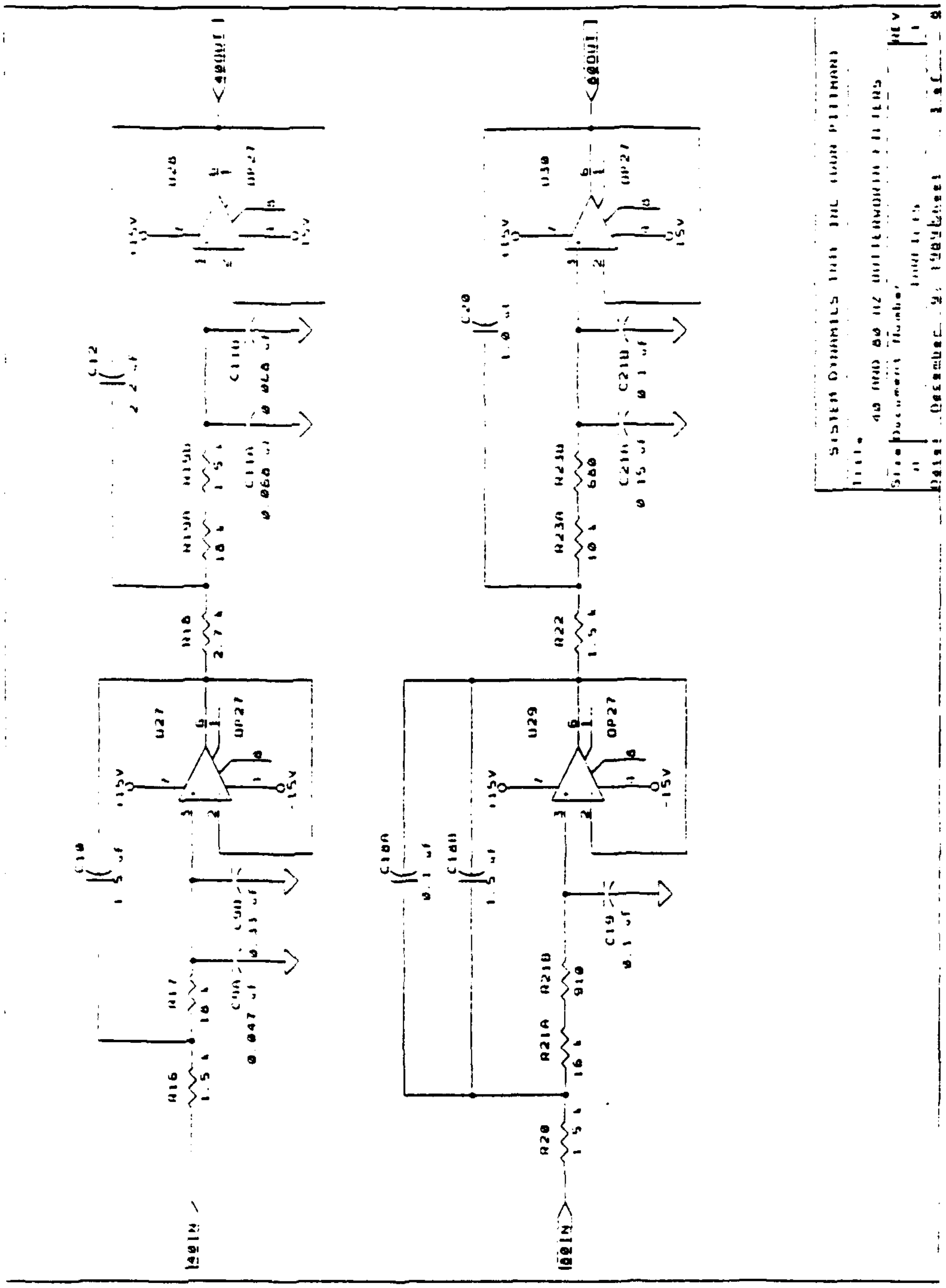


Figure 14 - 4-Pole Butterworth Filters

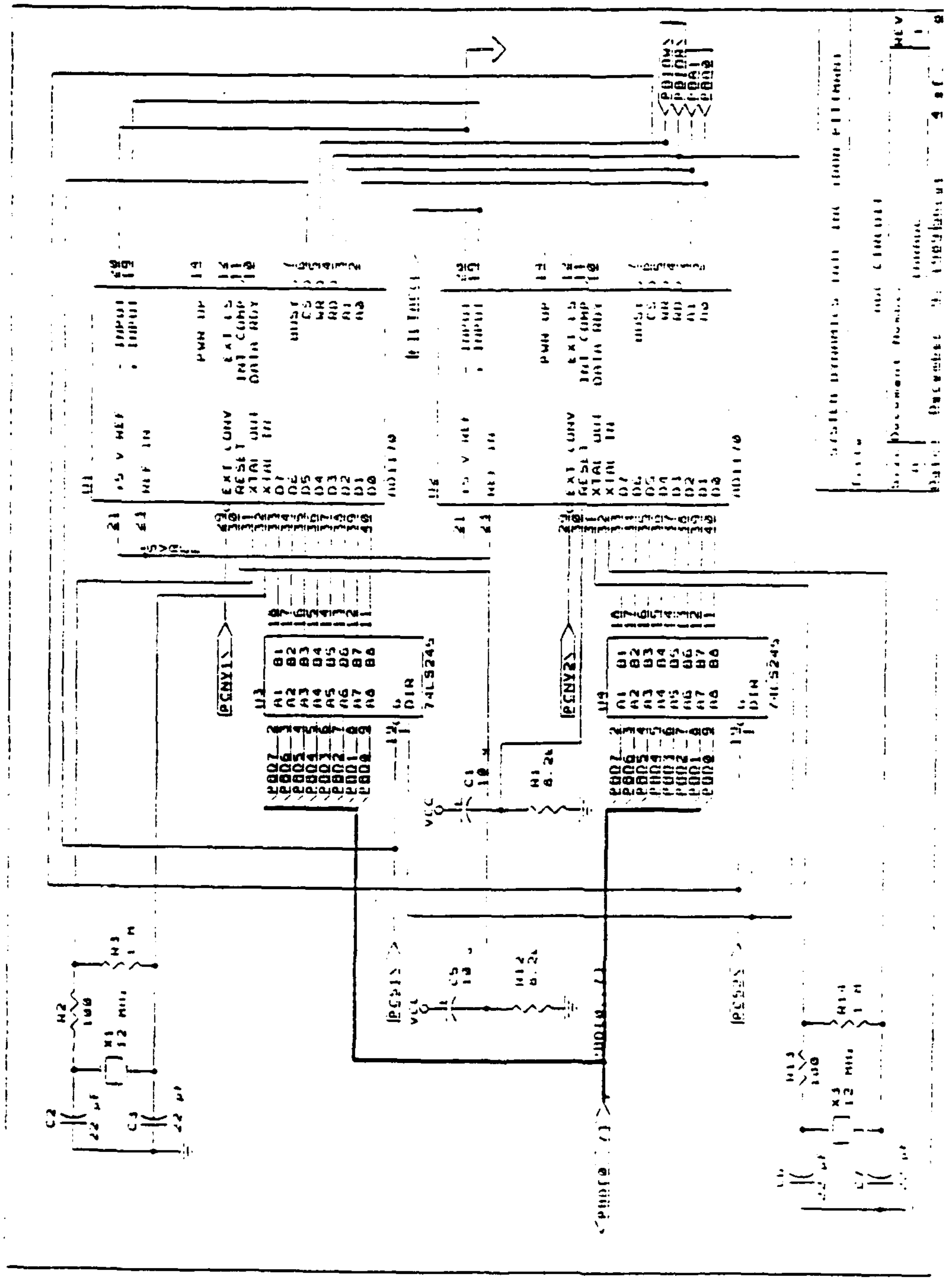


Figure 15 - AD1170 Interface Circuit

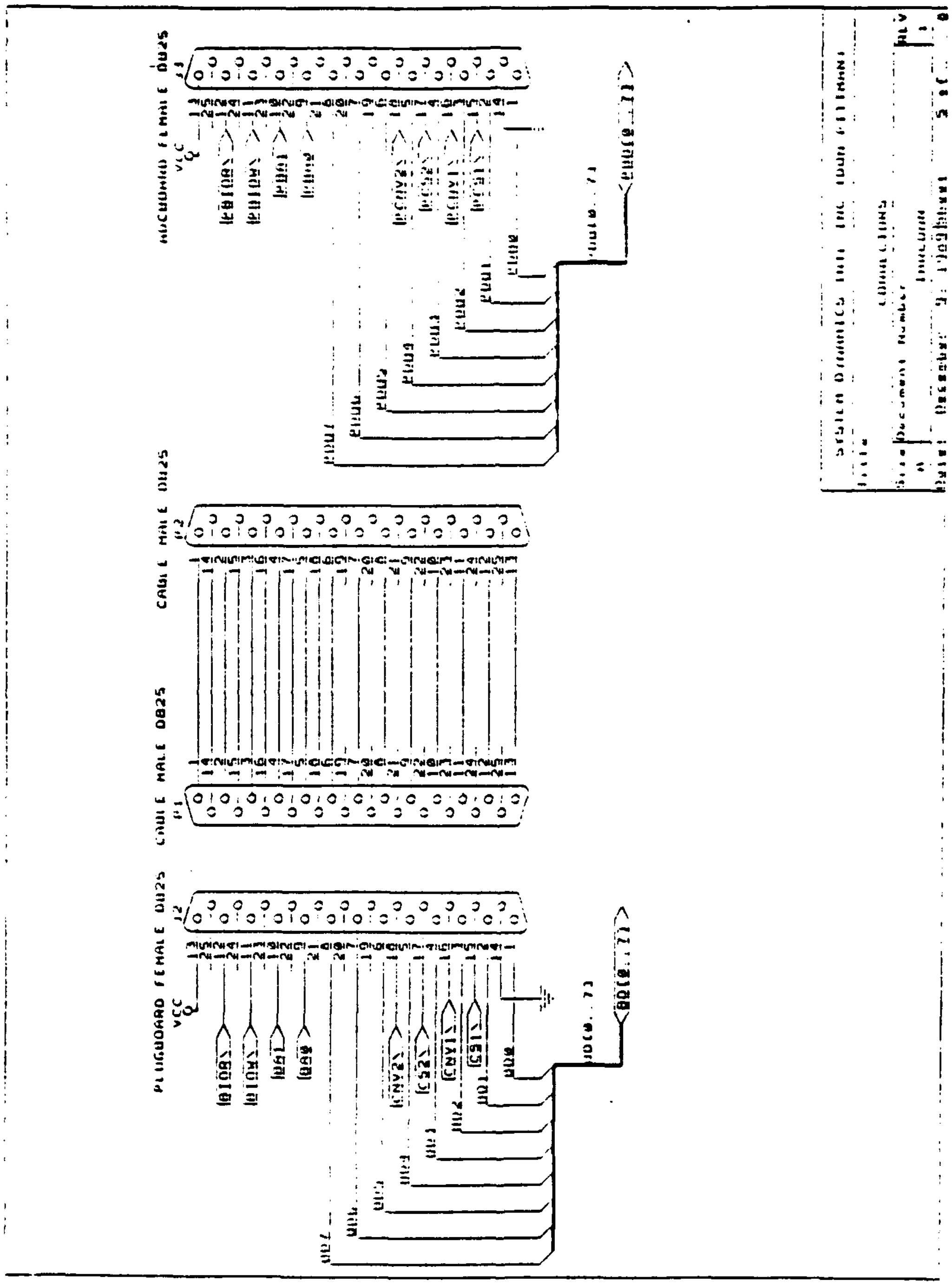


Figure 16 - DAS Cable Pinout

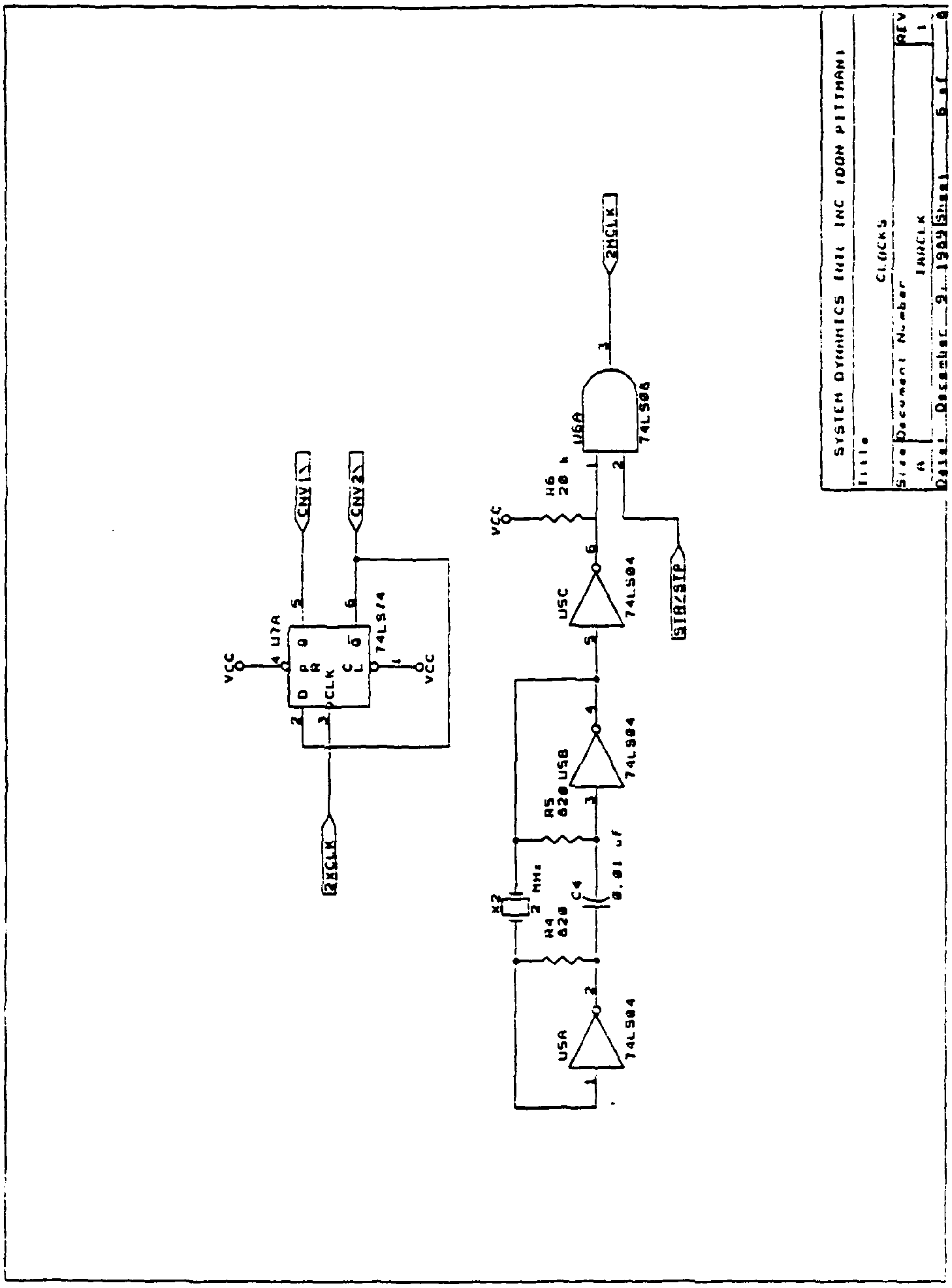


Figure 17 - 2 MHz Clock Circuit and Conversion Trigger Clock

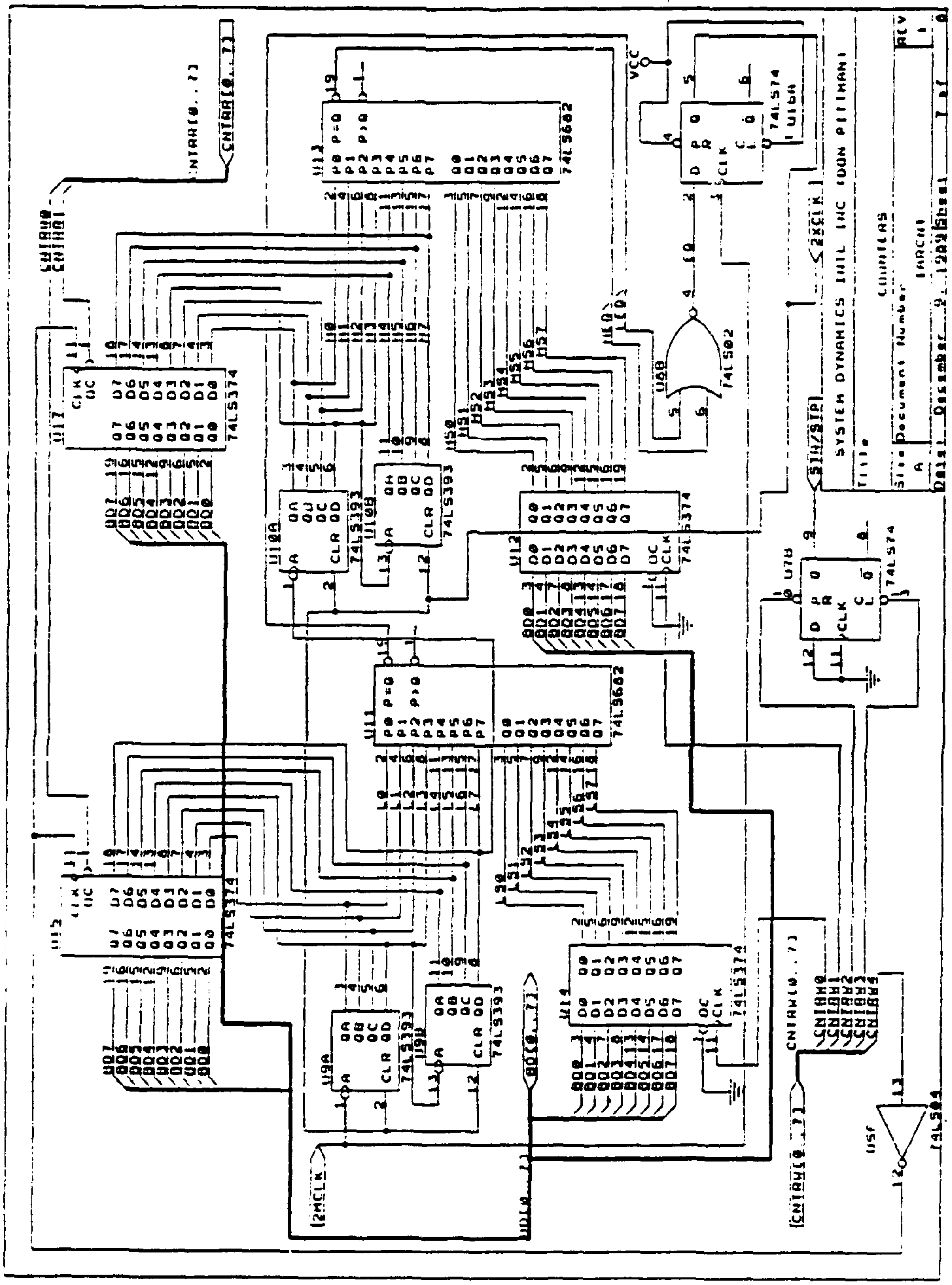


Figure 18 - Counter Timer Circuit

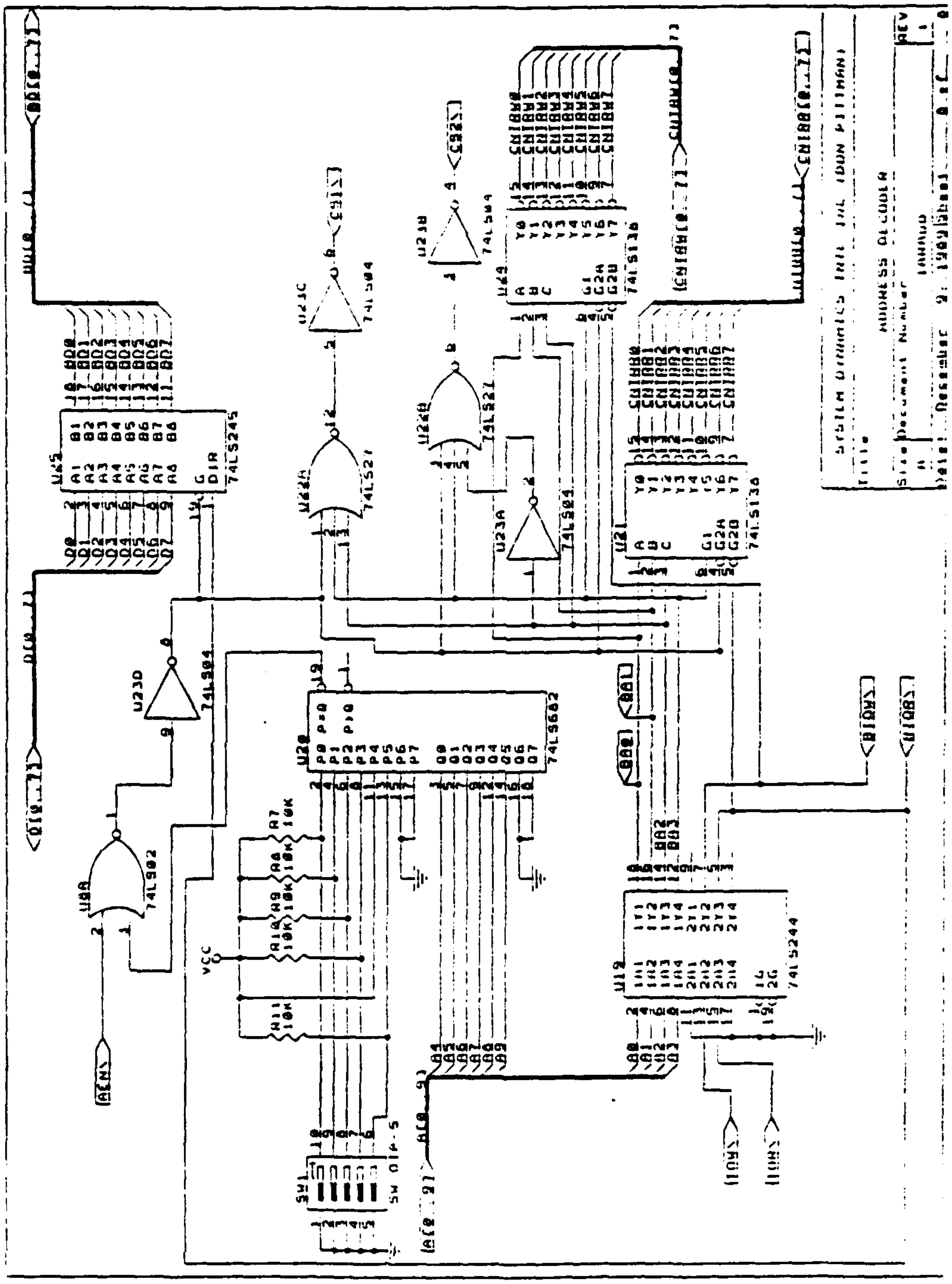


Figure 19 - Address Decoder Circuit

Table 2 - Jumper Selections for Different Conversion Ranges

Range	JMP1	JMP4	JMP7
± 1 G	2-4	2-1	1-3
± 1/2 G	3-4	3-1	1-3
± 1 G	2-4	2-1	1-2
± 1/2 G	3-4	3-1	1-2

Table 3 - Jumper Selections for Different Gravity Components

Gravity Component	Range	JMP2	JMP3
+ 1	± 1 G	X	6-8
+ 1	+ 1/2 G	6-8	X
0	+ 1 G	X	5-6
0	+ 1/2 G	5-6	X
- 1	+ 1 G	X	4-6
- 1	+ 1/2 G	4-6	X

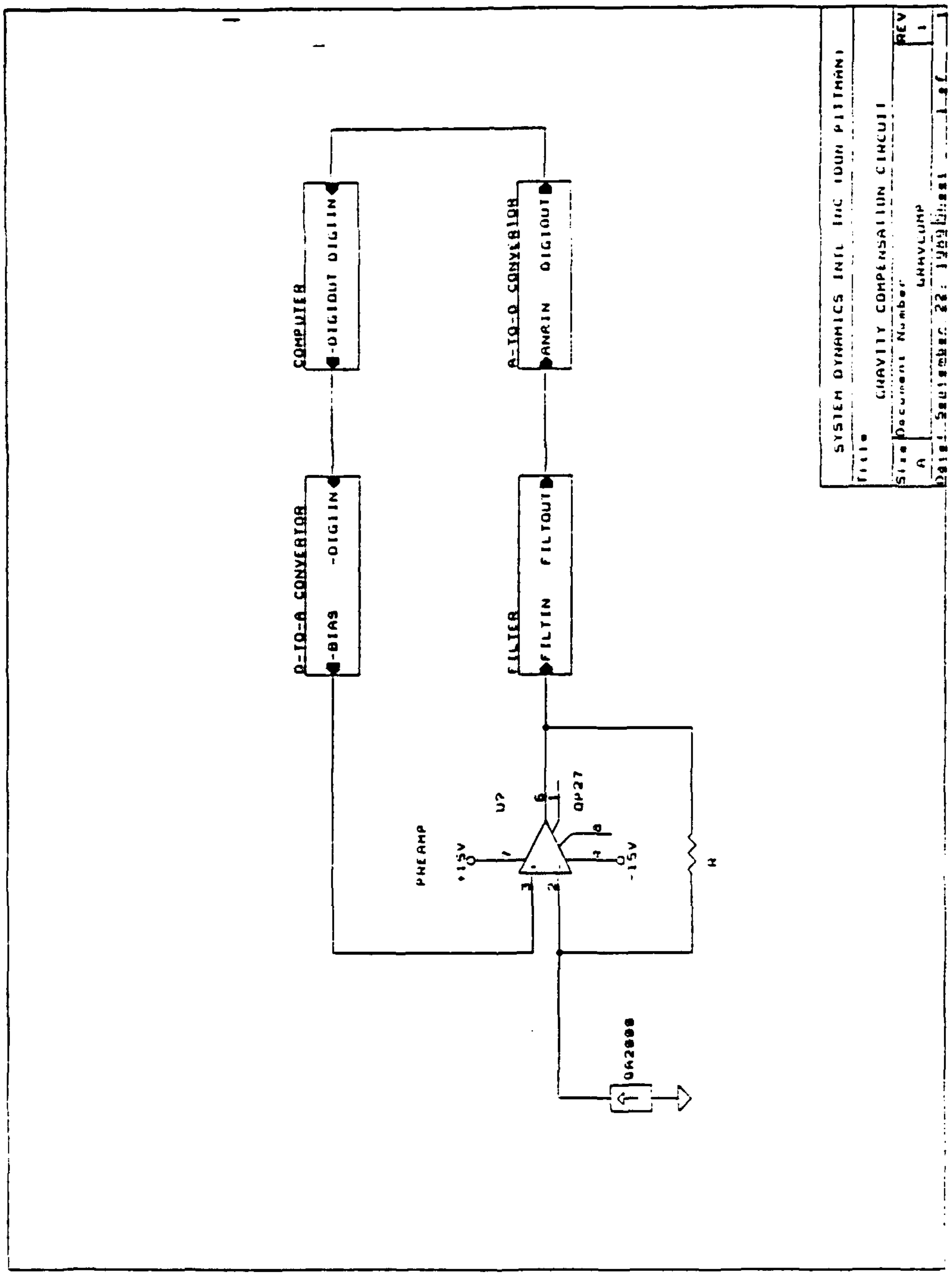


Figure 20 - Gravity Compensation Scheme

Two 4-pole butterworth filters (Figure 14) with bandwidths of 40 Hz and 80 Hz were used to filter out the noise components of the QA2000 above the folding frequency. A magnitude response plot for each filter is pictured in Figure 21 and 22. The two bandwidths are jumper selectable as shown in Table 4.

Table 4 - Jumper Selections for Filter Bandwidth

Filter BW	JMP5	JMP6
40 Hz	1-2	1-2
80 Hz	1-3	1-3

In a digital sampling system the folding frequency is defined as being half the sampling frequency (f_s). Any frequency components of the signal being sampled that are above the folding frequency get "folded" down into the range between dc and $f_s/2$. This phenomenon is also known as "aliasing". Therefore to reduce the effects of aliasing, filters are placed before the A-to-D convertor. The selection of the filter bandwidths used in the DAS (40 Hz and 80 Hz) were selected for nominal sampling frequencies of 100 Hz and 200 Hz.

A Sallen-Key active filter configuration was used to implement the filters discussed here. To insure that the filter elements would not add any noise comparable to that of the accelerometer, a noise analysis was conducted prior to fabrication. This analysis was done for a 2-pole filter without a preamp as shown in Figure 23. The input to output transfer function for this arrangement is

$$\frac{V_{out}}{I_{in}} = \frac{R}{1 + sC_2(R + R_1 + R_2) + s^2R_2C_2C_1(R + R_1)}$$

The current sources (except for the input source) shown in the figure are actually white noise current power models for the resistors and the opamp having units of amps^2/Hz. A simple nodal analysis results in the following transfer functions for each noise source.

For R_1

$$\frac{V_{out}}{I_{R1}} = \frac{-R_1}{R}\frac{V_{out}}{I_{in}}$$

For R_2

$$\frac{V_{out}}{I_{R2}} = \frac{-R_2(1 + sC_1(R + R_1))}{R}\frac{V_{out}}{I_{in}}$$

For R

$$\frac{V_{out}}{I_R} = \frac{V_{out}}{I_{in}}$$

For E_N

$$\frac{V_{out}}{E_N} = 1$$

For I_{N+}

$$\frac{V_{out}}{I_{N+}} = \frac{(R + R_1 + R_2)(1 + sC_1(R_2//(R + R_1))}{R} \frac{V_{out}}{I_{in}}$$

Replacing the resistor noise currents by 4kT/R where k is Boltzmann's constant (1.38×10^{-23} J/^{o}K) and T is temperature in ^{o}K (T_{room} = 300 ^{o}K), results in the following equations for the total input referred noise power,

$$\overline{I_{in}^2} = 4kT \left[\frac{R + R_1 + R_2(1 + sC_1(R + R_1))^2}{R^2} \right]$$

$$+ \overline{I_{N+}^2} \left[\frac{(R + R_1 + R_2)^2}{R^2} (1 + sC_1(R_2//(R + R_1))^2 \right]$$

$$+ \overline{E_N^2} \left[\frac{I_{in}}{V_{out}} \right]^2$$

and the output referred noise power,

$$\overline{E_o^2} = \overline{I_{in}^2} \left[\frac{V_{out}}{I_{in}} \right]^2 .$$

Substituting in the appropriate values for an 80 Hz Butterworth filter (Table 5) in the equation for the total output power and then integrating from dc-infinity and taking the square root gives an rms output referred noise voltage of 1.0 μV. Referring this value to the input and multiplying by the QA2000 scale factor (1.25 mA/G) gives the input referred noise acceleration of 0.1 μG, which is negligible compared to the accelerometer noise 1200 μG.

The noise components of the QA2000 are shown in Figures 10 and 11 of the previous section. Figures 24 - 27 show the power spectrum of the accelerometer after being filtered. The 40 Hz and 80 Hz filters reduced the RMS noise of the QA2000 from 1200 μG to 75 μG and 85 μG respectively. The OP27 opamp from Precision Monolithics was used in all of the filter and amplifier designs because of its low noise characteristics (3 nV/$\sqrt{}$Hz) and bias stability (0.4

μV/month).

Table 5 - 2-Pole 80 Hz Butterworth Filter Elements

Element	Value
R	8.0 kΩ
R_1	5.9 kΩ
R_2	2.4 kΩ
C_1	0.690 μF
C_2	0.173 μF

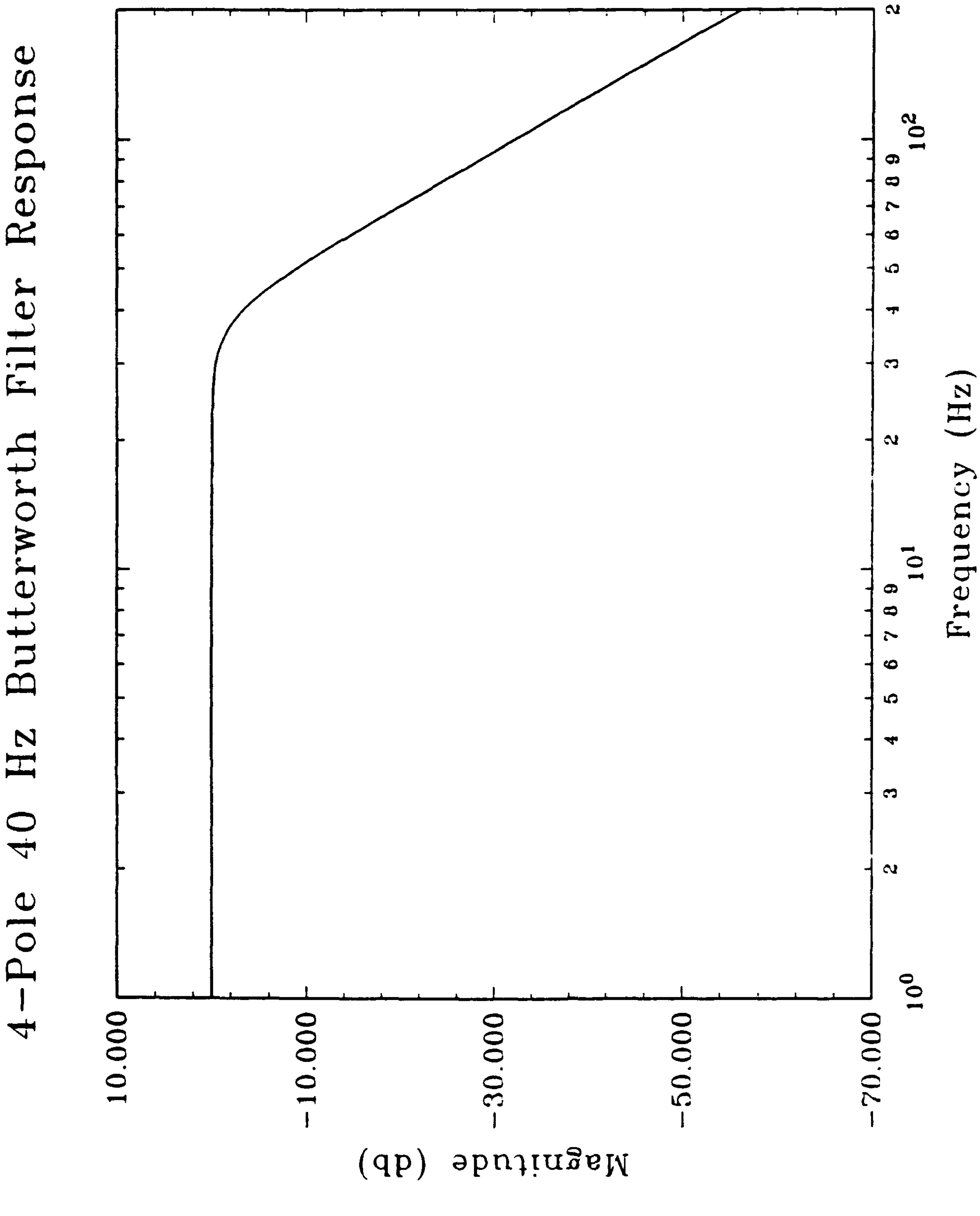


Figure 21 - Magnitude Response of a 40 Hz 4-pole Butterworth Filter

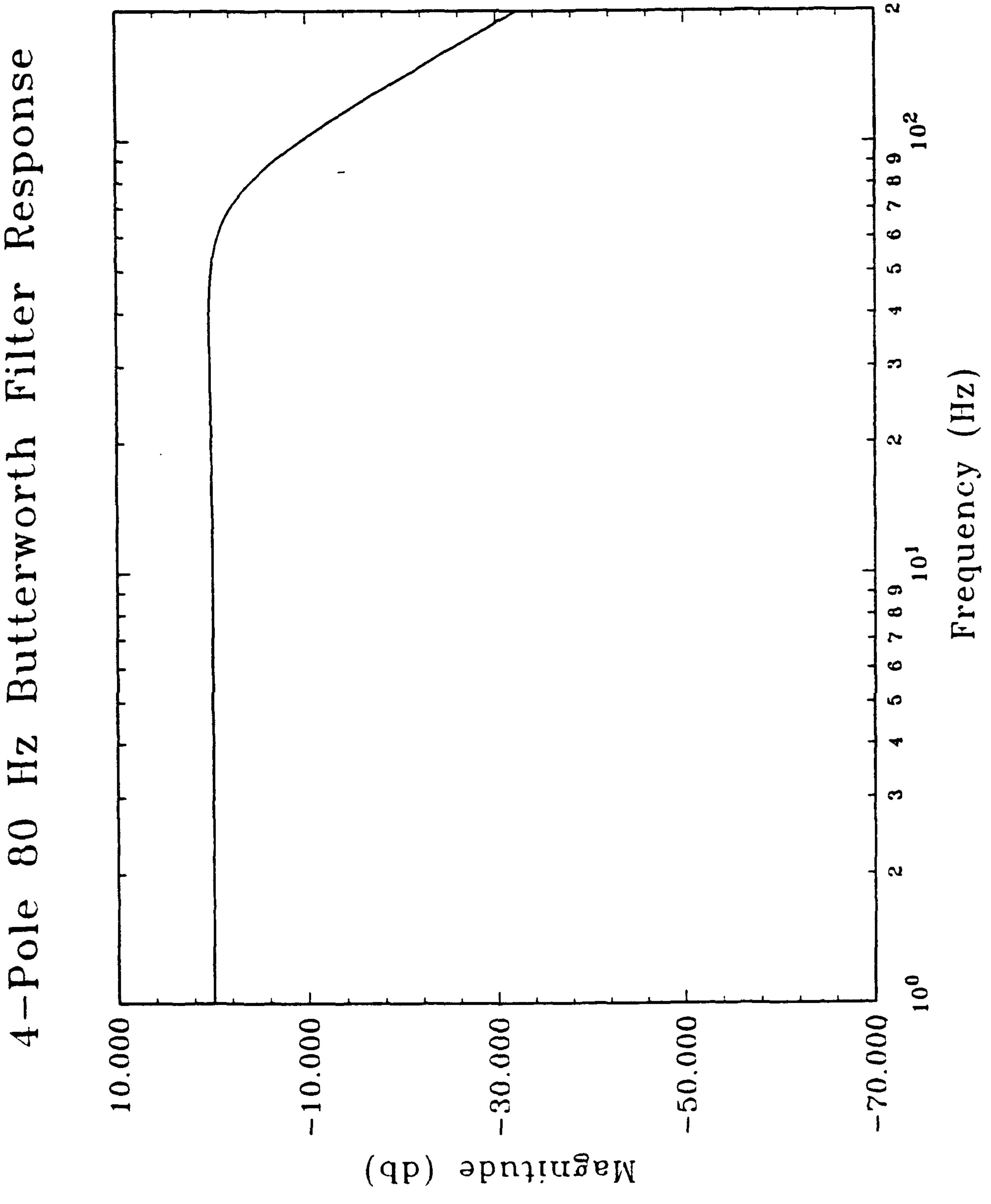


Figure 22 - Magnitude Response of an 80 Hz 4-pole Butterworth Filter

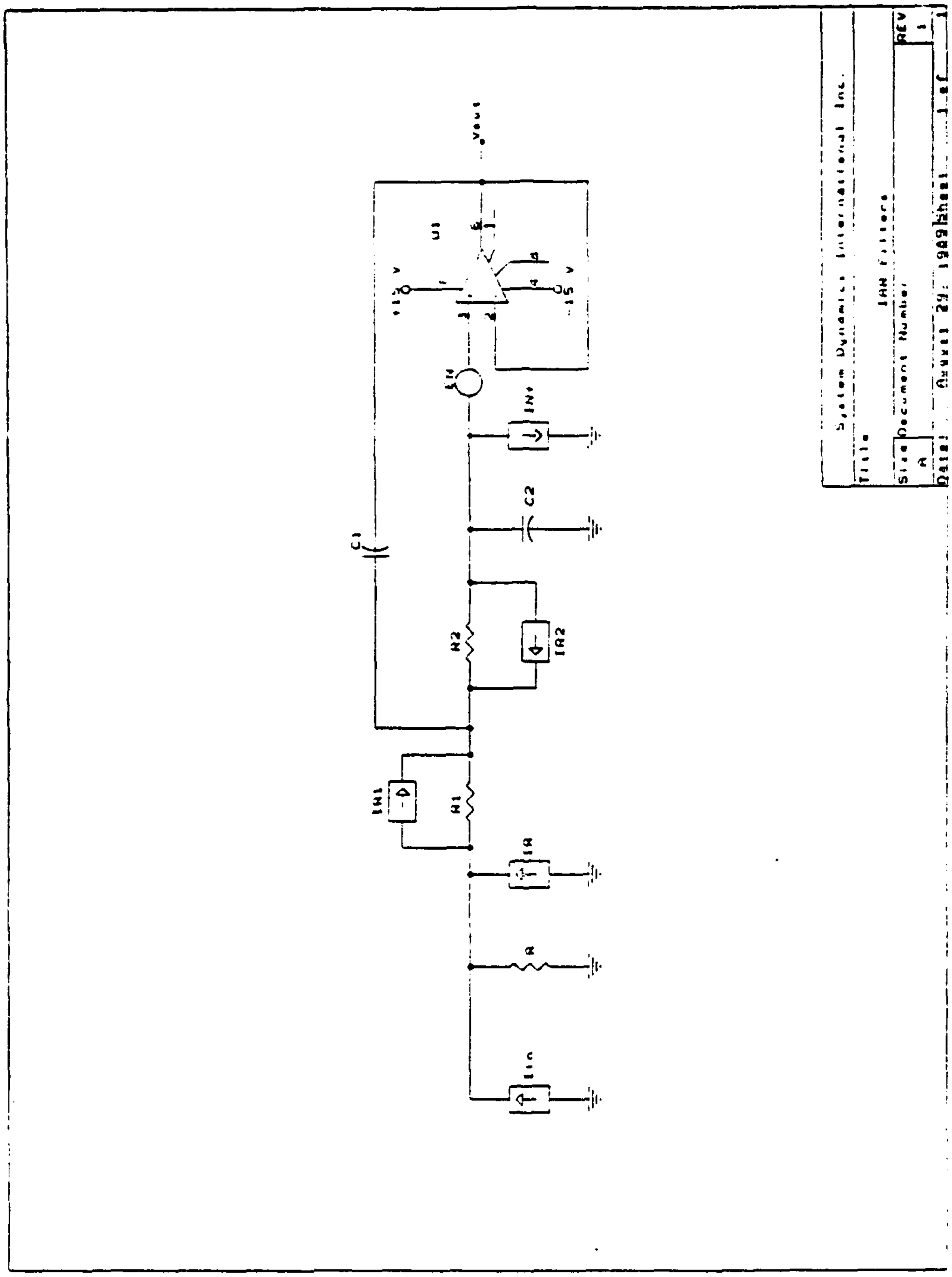


Figure 23 - Noise Analysis of a 2-pole Active Filter

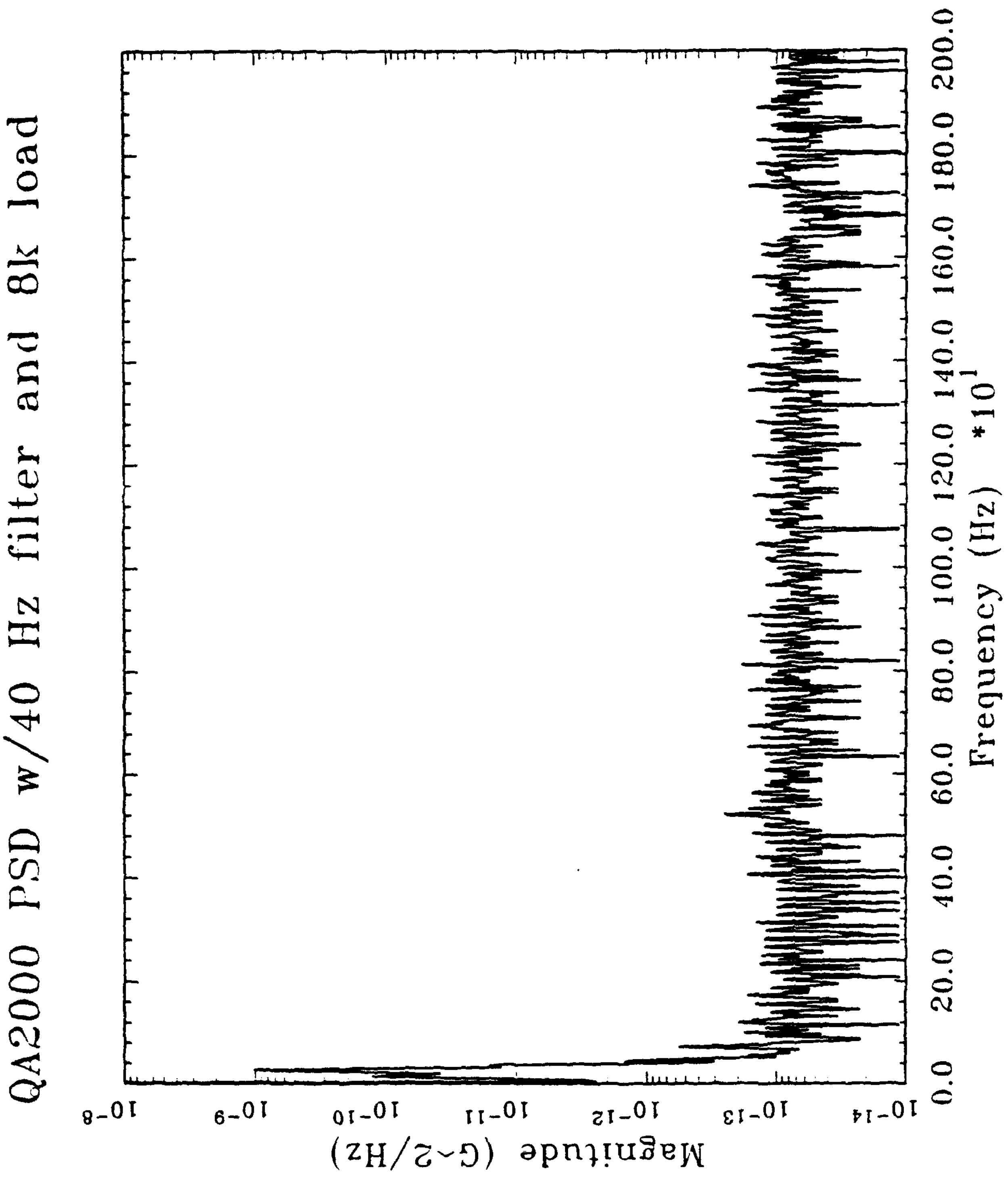


Figure 24 - QA2000 Power Spectrum Filtered by a 40 Hz 4-Pole Butterworth Filt?r (0-2kHz)

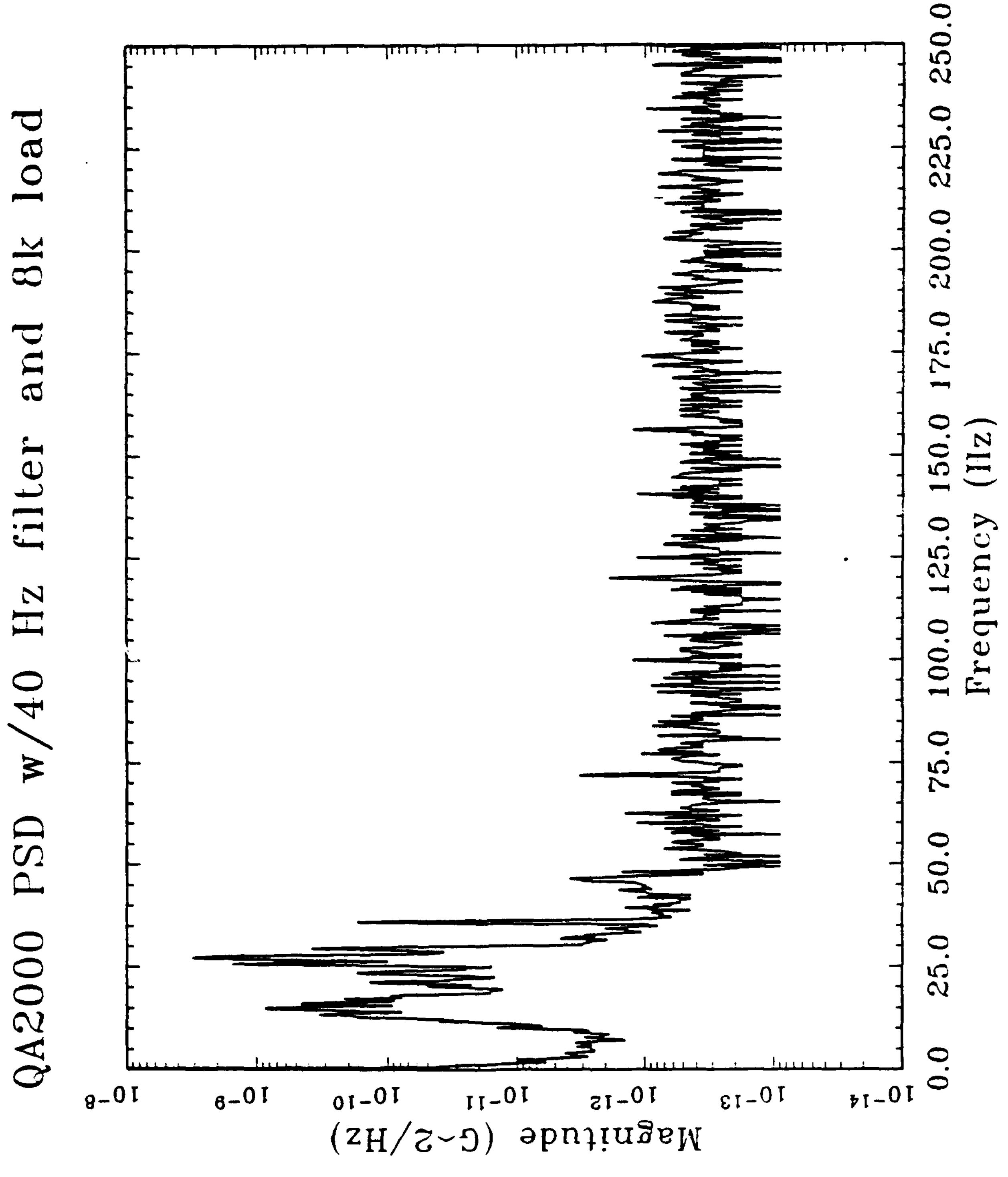


Figure 25 - QA2000 Power Spectrum Filtered by a 40 Hz 4-Pole Butterworth Filter (0-250 Hz)

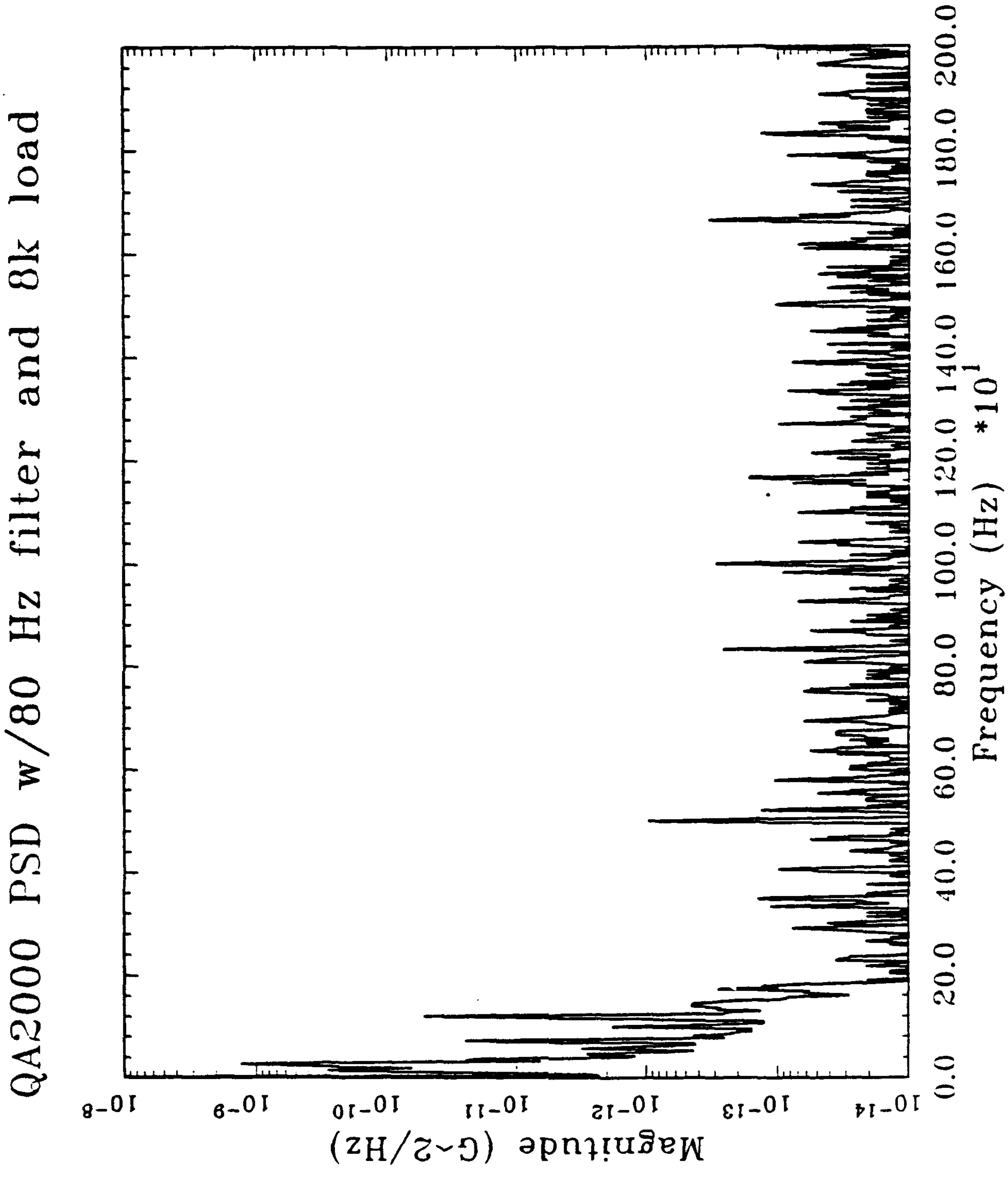


Figure 26 - QA2000 Power Spectrum Filtered by a 80 Hz 4-Pole Butterworth Filter (0-2kHz)

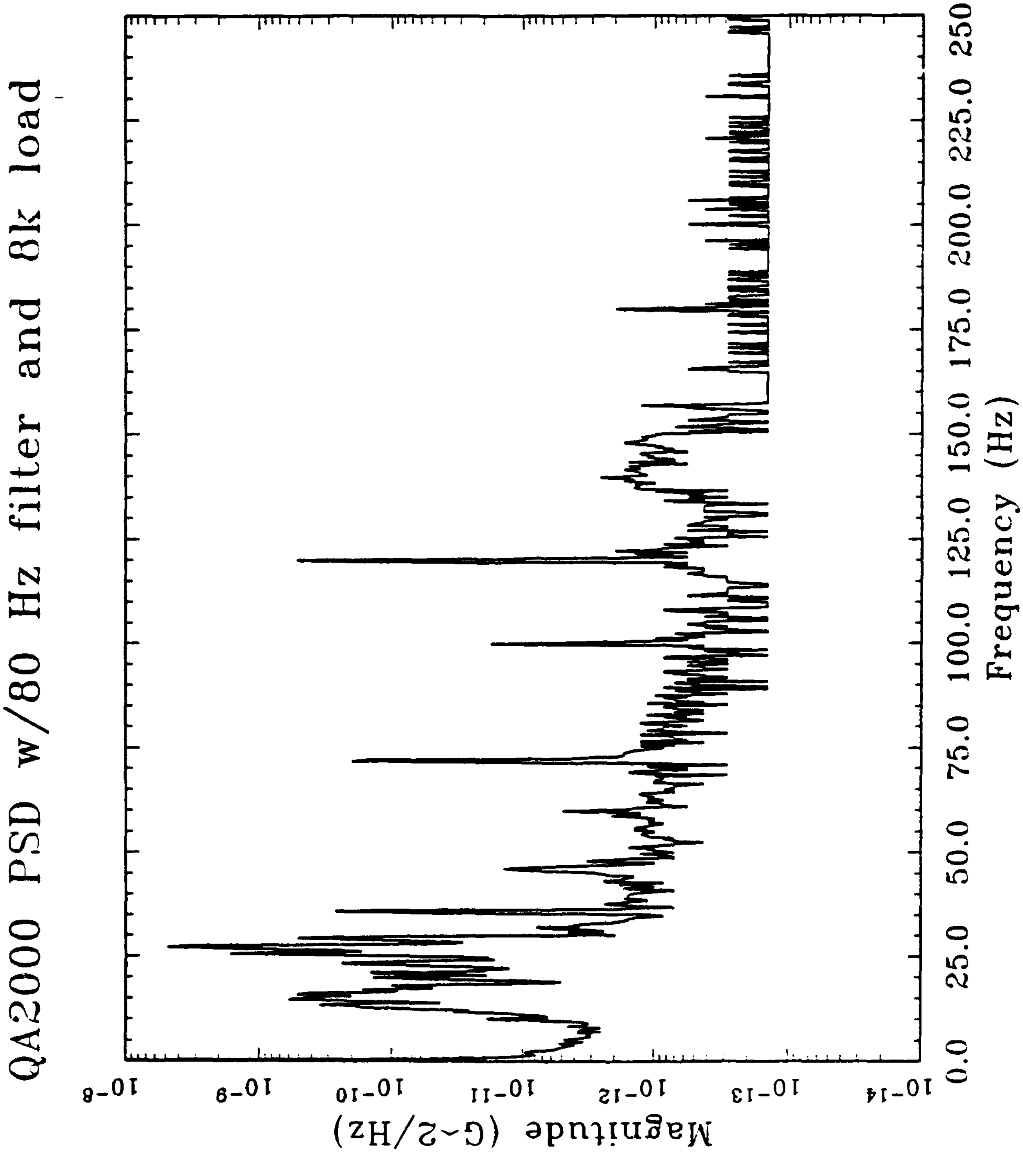


Figure 27 - QA2000 Power Spectrum Filtered by a 80 Hz 4-Pole Butterworth Filter (0-250 Hz)

4.3 AD1170 Interface Circuit

The interface to the AD1170's shown in Figure 15 was designed according to the requirements as specified in the data sheets for these devices. A copy of the AD1170 data sheet is included in Appendix A. The only deviation from the data sheet was the connection of the +5V reference of U1 to both reference inputs of each device. This change allowed the convertors to calibrate using the same reference which should increase the relative accuracy of the system. The input impedence of the XTAL pins and the RESET pin did not allow for combining the crystal oscillator circuits or the reset circuits. The external conversion signals, data lines, address lines, and the read and write strobes were all transmitted via a DB25 connector and cable (Figure 16) from the address decoder and counter timer circuits.

4.4 Programmable Counter Circuit

The programmable counter/timer (C/T) consists of a 2 MHz crystal oscillator, 16-bit load register, 16-bit counter, 16-bit counter latch, and a 16-bit comparator. The outputs of the circuit are two clock signals, 180^{o} out of phase, which are used to trigger the start of conversion for the two AD1170's.

To program the C/T for a DAS sampling frequency of f_s the low and high bytes of the result,

$$\text{Load value} = 2.0\text{x}10^{6}/f_s\ ,$$

are written to the 16-bit load register (U12 and U14) via the I/O bus. The following "c" subroutine can be used to program the counter for any sampling rate.

```
#define CNTR_LOW_BYTE  BASE+8
#define CNTR_HIGH_BYTE BASE+9

double setup_counter (double frequency)
  {
    unsigned int
        numb_counts,  chigh,  clow;

    double
        actual_freq;


    numb_counts = (unsigned int) (2.0E+06/frequency);
    chigh = (numb_counts >> 8) & 0x00ff;
    clow = numb_counts & 0x00ff;

    outp (CNTR_LOW_BYTE, clow);
    outp (CNTR_HIGH_BYTE, chigh);

    actual_freq = 2.0E+06/((double) numb_counts);
    return (actual_freq);
  }
```

After the registers are loaded and the start command (CNTRW2) is given, the 2 MHz clock (2MCLK) will start incrementing the counters (U9 and U10). The outputs of the two 8-bit comparators (U11 and U13) signal whenever the count on the counters is equal to the value set in the load register. The active low outputs of the comparators are "NORed" to provide an active high input to an edge-triggered D flip-flop (U16A) which outputs a 50% duty cycle clock (2XCLK) at the programmed sampling rate. This clock is then divided by 2, using another D flip-flop (U7A), to provide the conversion trigger clocks CNV1\ and CNV2\. Figure 28 shows the relative timing of these signals as measured by a logic analyzer. Once this process is completed then the data ready bit of one AD1170 can be monitored until it signals the end of conversion. After reading this AD1170 then the program can begin to monitor the other AD1170. The following sample "c" program illustrates this procedure. The source code of the subroutines used in this program are included in the real-time position determination program in Appendix C.

```
#include <graph.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <bios.h>
#include "iarinc.h"

void
    setup_ADC_defaults (void),
    main (void),
    wait_for_A (void),
    wait_for_B (void);

double
    read_A (void),
    read_B (void),
    setup_ADC_int_time (double frequency),
    setup_counter (double frequency);


void main ()
  {
    char
        file_name[80];

    int
        i;

    double
        sample,  samp_freq,  int_freq;

    FILE
        *file_ptr;
/**********************************************************/
/* Open output file                                       */
/**********************************************************/

    printf ("\nEnter name of data file: ");
    gets (file_name);

    file_ptr = fopen (file_name, "w");
    if (file_ptr == NULL)
      {
        printf ("\n\n\t** Could not open %s for writing **",
                file_name);
        exit(0);
      }


/**********************************************************/
/* Program ADC and counter/timer for sampling rate of     */
/* 200 Hz                                                 */
/**********************************************************/

    outp (CNTR_STOP, 0x0000);            /* Disarm 2MCLK  */

    setup_ADC_defaults ();

    samp_freq = setup_counter (200.0);

    int_freq = setup_ADC_int_time (samp_freq);

    outp (CNTR_START, 0x0000);            /* Arm 2MCLK  */


/**********************************************************/
/*  Loop until 2048 samples have been taken               */
/**********************************************************/

    for (i = 0; i < 1024; ++i)
      {
        sample = read_A ();
        fprintf (file_ptr, "\n%g", sample);

        sample = read_B ();
        fprintf (file_ptr, "\n%g", sample);
      }
  }
```

The advantage of the hardware initiated conversion is that the processor can be used for other task besides controlling the A-to-D convertors and the timing jitter is minimized.

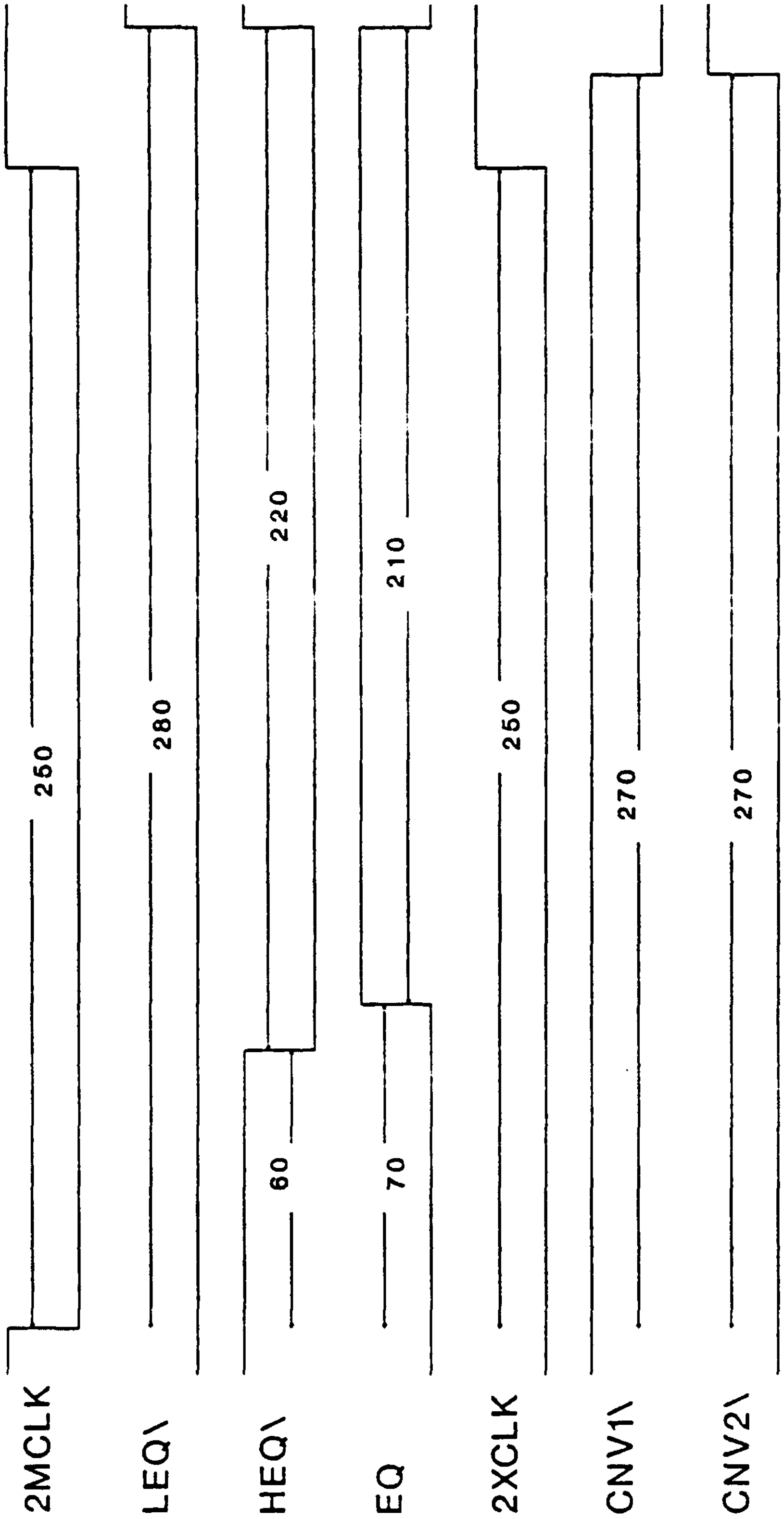


Figure 28 - Counter/Timer Circuit Timing Diagram

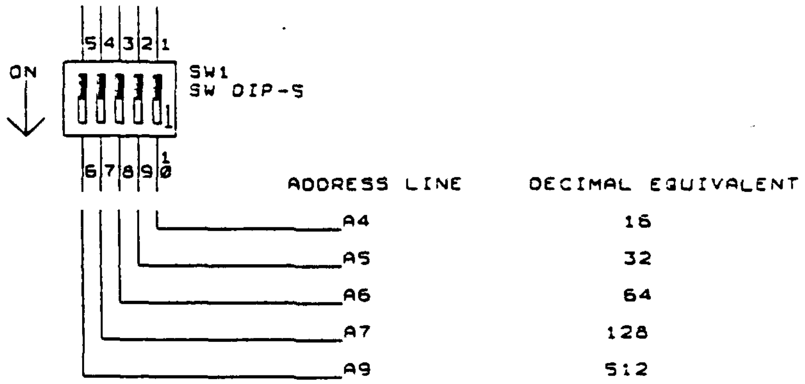
4.5 Address Decoder Circuit

The basic functions of the address decoder circuit shown in Figure 19 were to furnish chip select signals for the two AD1170's, provide individual read and write signals for the addresses occupied by the counter/timer, and buffer the data transfers to and from the bus. Connections to the IBM bus were made such that only one TTL LS load needed to be driven by the bus. Therefore the I/O read and write strobes, and the address bits A0-3 were buffered through a 74LS244.

The switch (SW1) was used to set the I/O base address of the acquisition system. The range of I/O base addresses is from 100 to 3F0 (Hexidecimal). (That is address bit A8 is not selectable but hardwired to a "1"). Figure 29 shows the address definitions for the switch. The real-time software is setup to run with a base address of 380 Hex. Table 6 shows the address map of the acquisition system. The last three I/O locations were reserved for future enhancements.

Table 6 - DAS Address Map

Register/Command	Read	Write	Address	Data Lines Used?
AD#1 Command Reg		*	Base+0	Yes
AD#1 Param Reg 1		*	Base+1	Yes
AD#1 Param Reg 2		*	Base+2	Yes
AD#1 Status Reg	*		Base+0	Yes
AD#1 Low Byte	*		Base+1	Yes
AD#1 Mid Byte	*		Base+2	Yes
AD#1 High Byte	*		Base+3	Yes
AD#2 Command Reg		*	Base+4	Yes
AD#2 Param Reg 1		*	Base+5	Yes
AD#2 Param Reg 2		*	Base+6	Yes
AD#2 Status Reg	*		Base+4	Yes
AD#2 Low Byte	*		Base+5	Yes
AD#2 Mid Byte	*		Base+6	Yes
AD#2 High Byte	*		Base+7	Yes
Cntr Low Byte	*	*	Base+8	Yes
Cntr High Byte	*	*	Base+9	Yes
Cntr Start		*	Base+10	No
Cntr Stop		*	Base+11	No
Cntr Latch Cnt		*	Base+12	No
* Reserved *			Base+13	
* Reserved *			Base+14	
* Reserved *			Base+15	



ADDRESS LINE	DECIMAL EQUIVALENT
A4	16
A5	32
A6	64
A7	128
A9	512

THE BASE ADDRESS IS SET TO THE SUM OF DECIMAL EQUIVALENTS OF THE ADDRESS LINES THAT ARE IN THE OFF OR "O" POSITION.

Figure 29 - Switch Setting for I/O Base Address

The output of the 8-bit comparator (74LS682), which signaled when the address bits A4-A9 were equal to the base address set by the switch, was "NORed" with the address enable strobe (AEN\) and then inverted to provide the enable signal for the octal bidirectional bus transceiver (74LS245). (The original design used a 74LS688 which has a built in enable eliminating the need for U8A and U23D). The bus transceiver passed the data bits between the bus and the acquisition system. The direction of the data transfer was controlled by the buffered I/O read strobe (BIOR\).

Using the active low transceiver enable (BASEADD\) along with the buffered address bits BA2 and BA3, the chip selects for the AD1170's are generated according to the boolean equations

$$\text{CS1}\backslash = \overline{\overline{\text{BASEADD}\backslash + \text{BA2} + \text{BA3}}}$$

and

$$\text{CS2}\backslash = \overline{\overline{\text{BASEADD}\backslash + \overline{\text{BA2}} + \text{BA3}}} \quad .$$

The lower 8 I/O locations, which are designated for use with the programmable counter/timer, are decoded using two 74LS138 decoders, one for reading and one for writing. The read (CNTRR0-7) and write (CNTRW0-7) selects are active low strobes that coincide with the I/O read and write strobes (BIOR\ and BIOW\).

4.6 Data Acquisition System Component Layout

The DAS was physically divided into two sections. To reduce errors due to external noise sources, the preamp, filters, and AD1170 interface were placed in a shielded enclosure outside of the main processing computer and the connection to the accelerometer output was via a shielded coaxial cable. The 25-pin cable was used to communicate with the address decoder and counter/timer circuits located inside the microcomputer. The component layouts for each section are pictured in Figures 30 and 31.

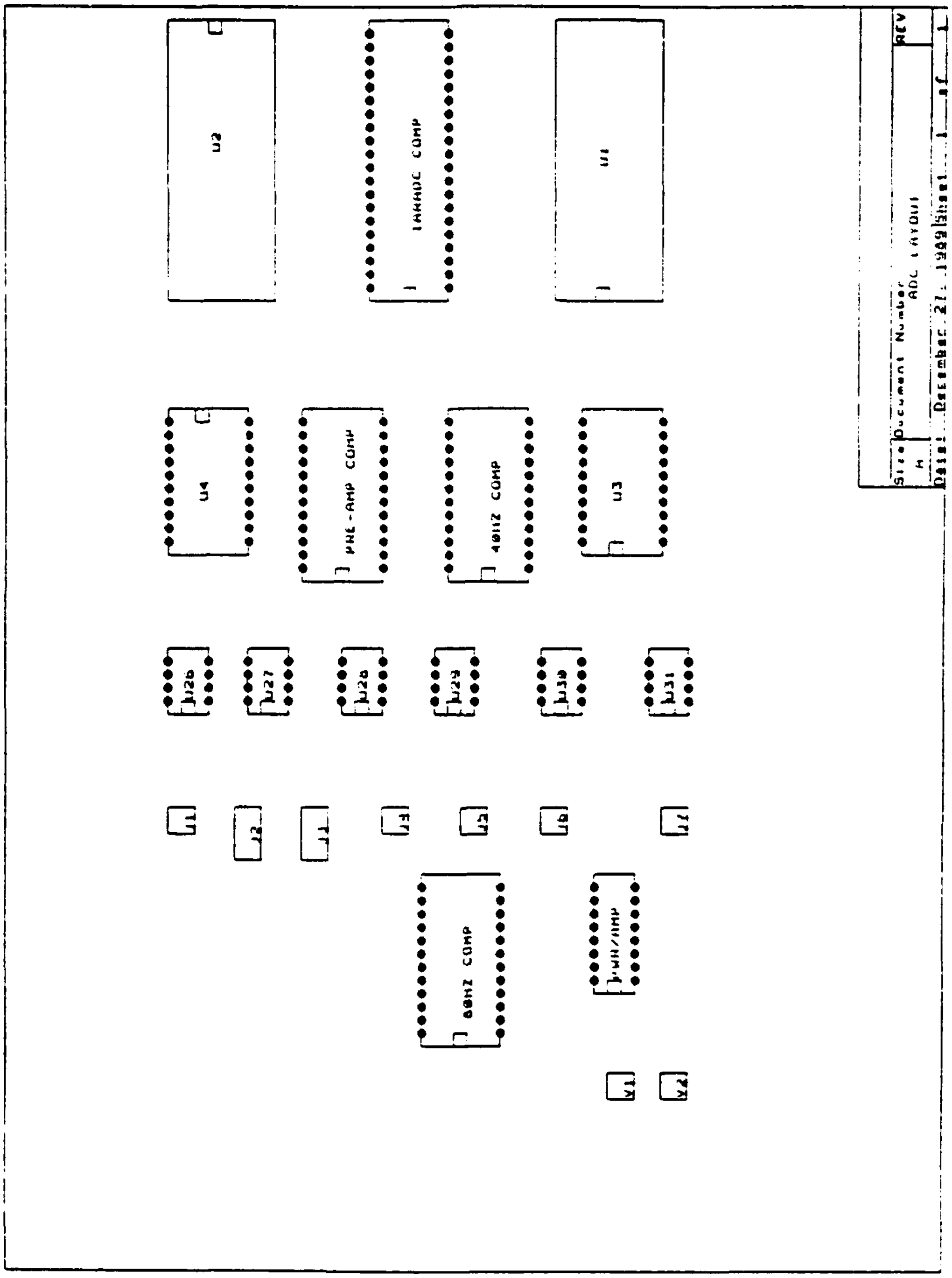


Figure 30 - Component Layout for Preamp, Filters, and AD1170 Interface

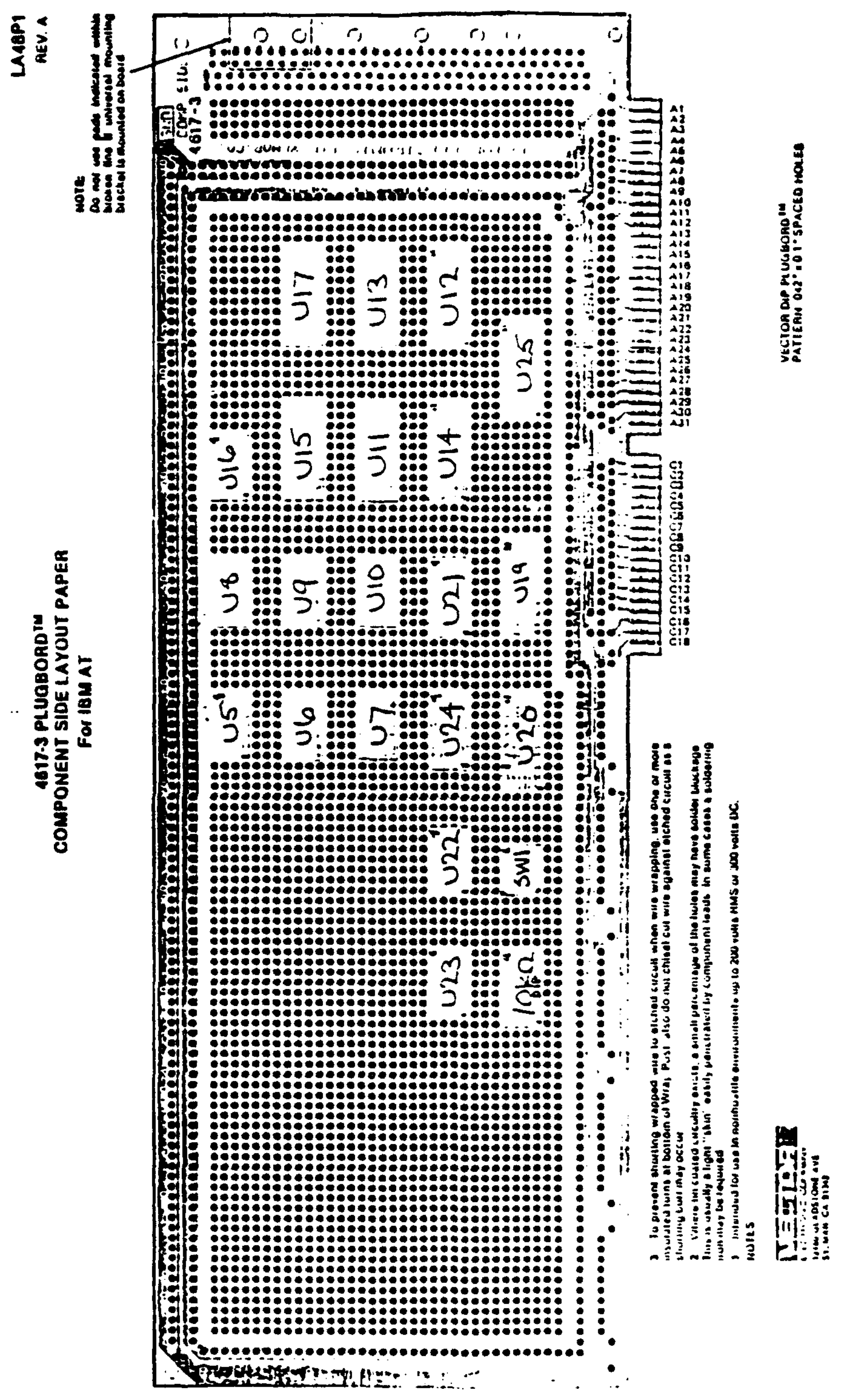

LA48P1
REV. A
NOTE:
Do not use pads indicated within broken line if universal mounting bracket is mounted on board
4617-3 PLUGBORD™
COMPONENT SIDE LAYOUT PAPER
For IBM AT
COMP. SID.
4617-3
U17
U13
U12
U25
U16
U15
U11
U14
U8
U9
U10
U21
U19
U5
U6
U7
U24
U26
U22
SW1
U23
10kΩ
VECTOR DIP PLUGBORD™
PATTERN .042" x .01" SPACED HOLES
NOTES

Figure 31 - Component Layout for Address Decoder and Counter/Timer

4.6 Data Acquisition System Software

During the development of the hardware and software for the real-time position determination system, several "mini" programs were written that are still useful in operating the DAS and Anorad table. These programs are listed in Table 7 with a brief description of the program's purpose. The source codes were delivered along with the system hardware and software and therefore are not included in this report.

Table 7 - DAS System Software List

Program Name	Description
ANORAD.BAS	User friendly environment for controlling the Anorad linear translation/reference table.
1170SAMP.C	DAS data collection program. Variable sampling rate. Up to 2048 data points.
1170ANO.C	Same as "1170SAMP" except it also reads the position of the Anorad table before and after sampling to determine total distance moved.
1170ANOB.C	Same as "1170ANO.C" except it also commands the Anorad table to move. The command can be changed in the program and recompiled for different movements.
ADCAL.C	Program used to calibrate the AD1170's to an external +5V reference.
ADSAVE.C	Program used to save all of the default parameters of the AD1170's to nonvolatile memory.
ADRECAL.C	Recalls the last set of saved AD1170 parameters.
AD1170.C	Uses the AD1170's as a simple digital voltmeter.
1170STOP.C	Stops the conversion signals in the DAS so that the "AD1170" program can be used.

Table 7 (cont.) - DAS System Software List

Program Name	Description
MULTTRAP.C	Processes off-line data taken from the accelerometer. Compatible with "1170ANO" and "1170ANOB". The program compensates for accel bias and then integrates twice to determine position. The first run in a set is used to determine the scale factor. The output of the program is a file containing the velocity error and position error for each run. The data files in one set should be name with the same first six letters followed by a number. Do not use an extension. ex/ 1RUN1, 1RUN2, 1RUN3,...1RUN9 This program integrates using the trapezoidal method.
MULTSIMP.C	Same as "MULTTRAP" except it uses the Simpson's rule for integration.
MULTADAM.C	Same as "MULTTRAP" except it uses the Adam-Basforth method for integration.

5 Affects of System Parameters on Position Error

5.1 Data Acquisition System Parameters

A computer simulation was conducted to determine the effects of the anti-aliasing filter bandwidth, the data sampling rate, and the resolution on position error. The simulation was programmed using a simulation environment called SimPack which was developed by System Dynamics [6]. SimPack allows one to simulate any system, linear or non-linear, analog or digital. A SimPack user develops the models he needs and then programs them in FORTRAN. The program used for the DAS system is included in Appendix B.

The inputs to the simulation were filter bandwidth, sampling rate, resolution, distanced moved, and simulation minimum and maximum rates. Based on the desired distance moved the program would calculate the appropriate time intervals of a predefined acceleration input. A typical acceleration input is shown Figure 32. The simulation did not model any accelerometer errors.

A typical plot of the position error versus time is provided in Figure 33. The error grows during positive acceleration reaching a maximum when the acceleration turns negative and returning near zero when the velocity returns to zero. The fact that the final error is negligible means that the errors of the DAS tend to cancel over a movement that starts at zero velocity and ends at zero velocity.

Each of the three parameters under consideration were swept independently over reasonable ranges. The two parameters that were not being swept for a given set were set to default values that would have negligible effects on the results as shown in Table 8. An exécetion was the case where the sampling rate was swept in which the filter bandwidth was always set to 1/4 of the sampling rate.

The results are presented in the form of plots where the final position error and the maximum position error are plotted versus the particular parameter being swept. Figures 34 - 36 show that the bandwidth and sampling rate have a negligible effect on the final position error while a resolution greater than 14 bits will result in a final error less than 1 thousandth of an inch. Figures 37 - 39 show that a resolution of at least 12 bits will reduce the maximum error down to 10 thousandths while a sampling rate of than 2 kHz and bandwidth of 1 kHz is required for the same maximum error.

The results of this simulation show that a DAS with a higher sampling rate (4 kHz) and lower resolution (16-bits) would be more appropriate than the low bandwidth (100 Hz) and high resolution (20-bits) required in this task.

Table 8 - Simulation Defaults

Parameter	Variable	Default
Filter Bandwidth	Fc	1 kHz
Sampling Rate	Samp_Rate	4 kHz
Resolution	Numb_of_Bits	32 bits

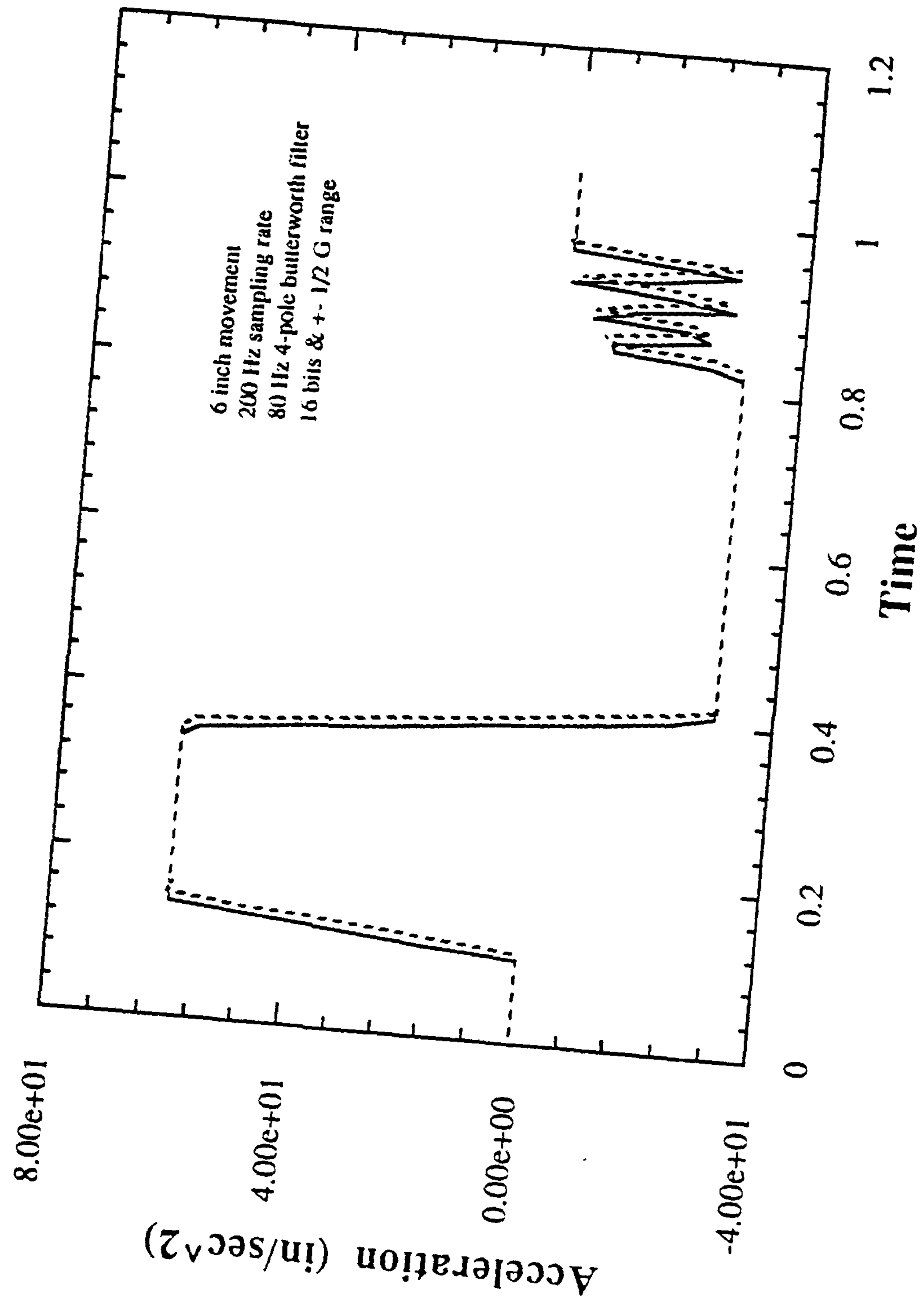


Figure 32 - Typical Acceleration Input

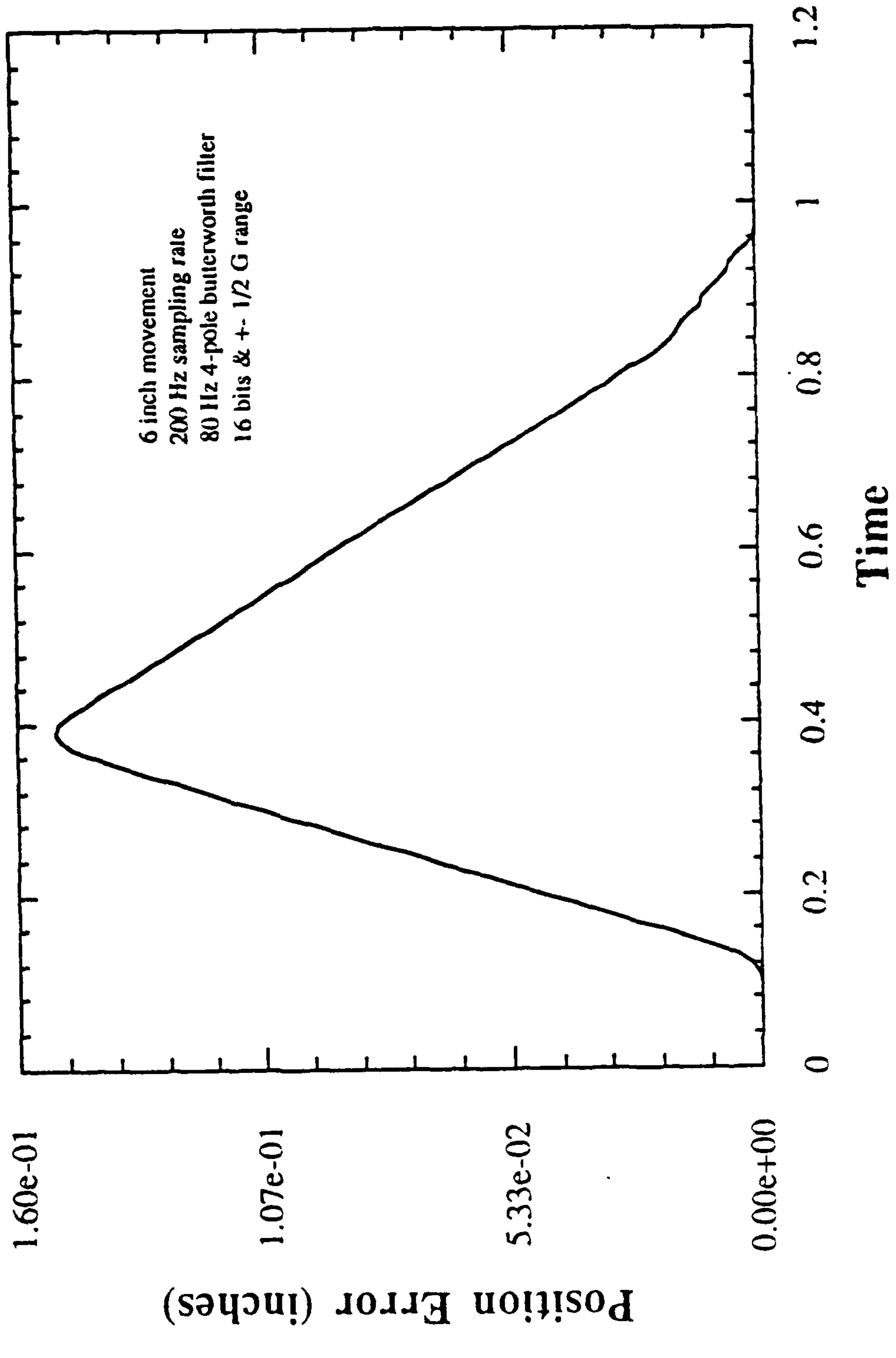


Figure 33 - A Typical Plot of Position Error versus Time

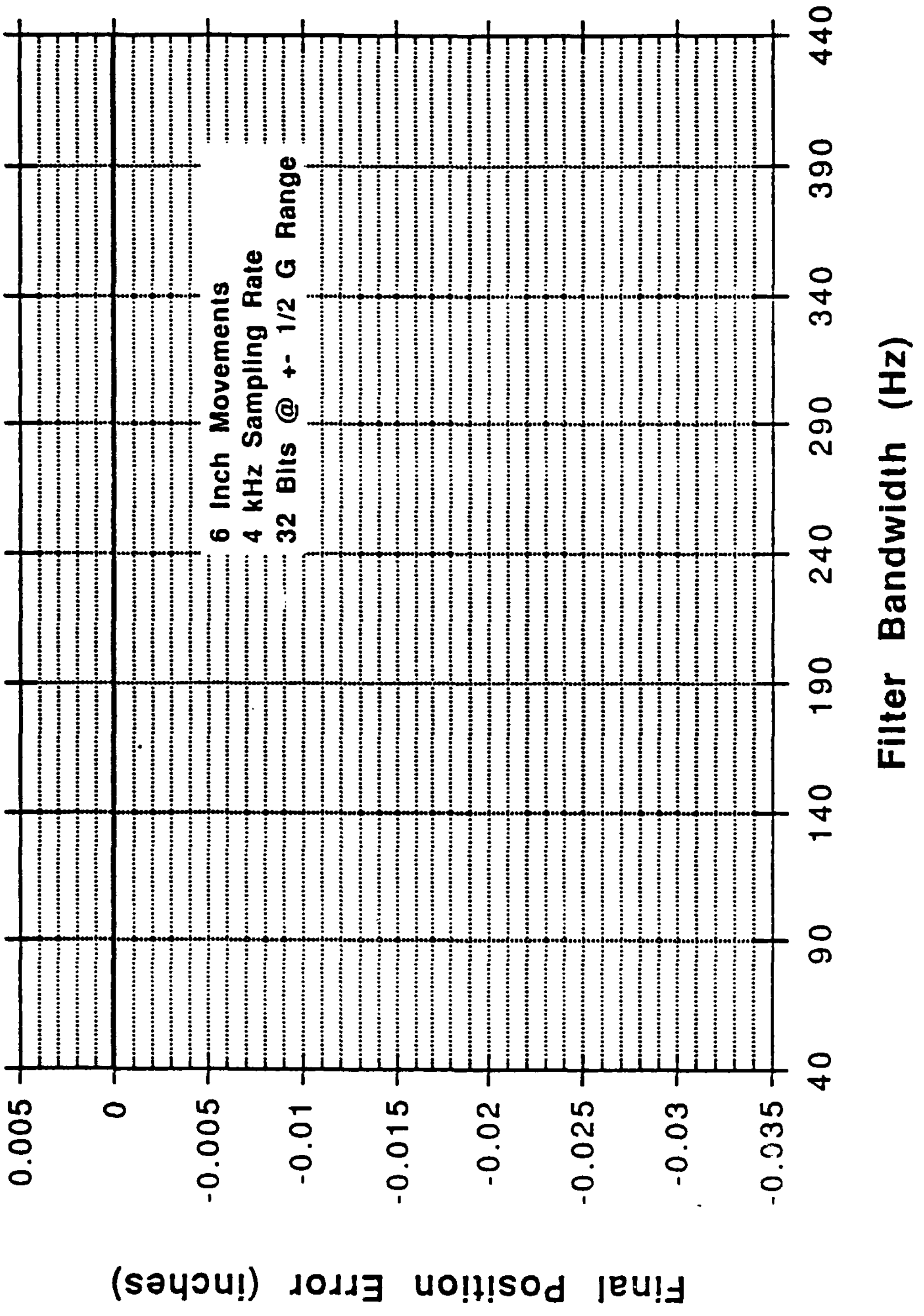


Figure 34 - Final Position Error versus Filter Bandwidth

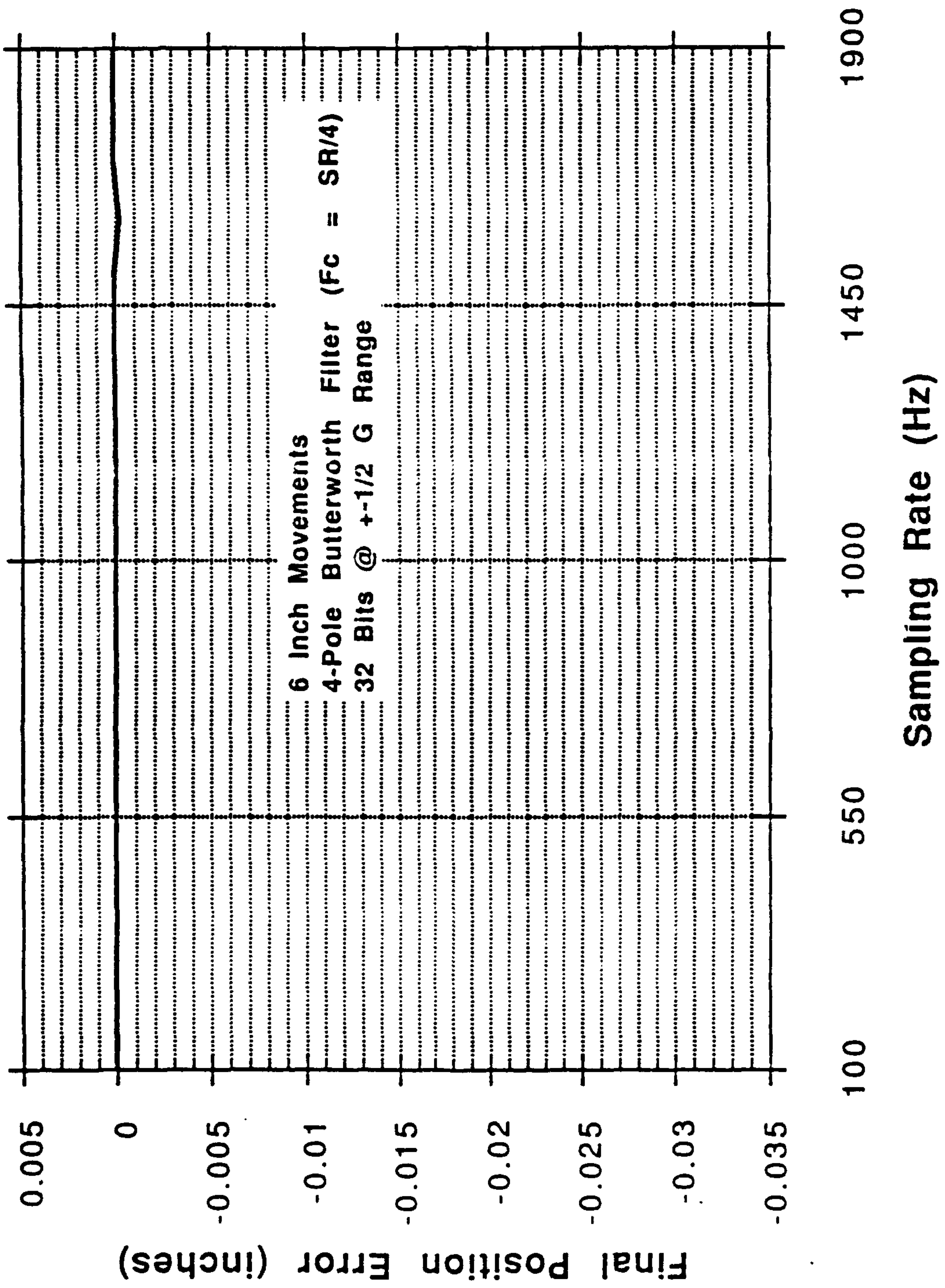


Figure 35 - Final Position Error versus DAS Sampling Rate

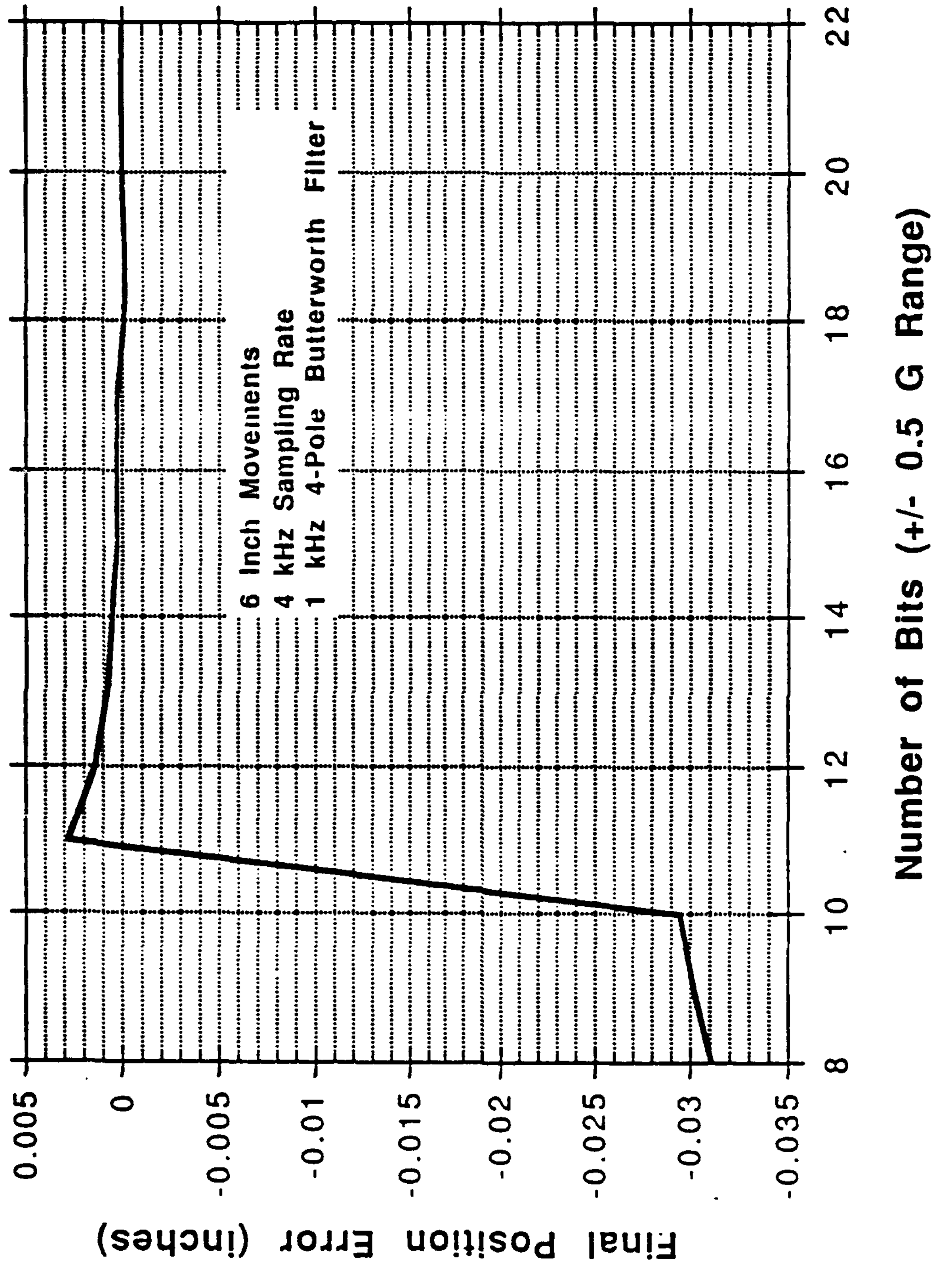


Figure 36 - Final Position Error versus DAS Resolution

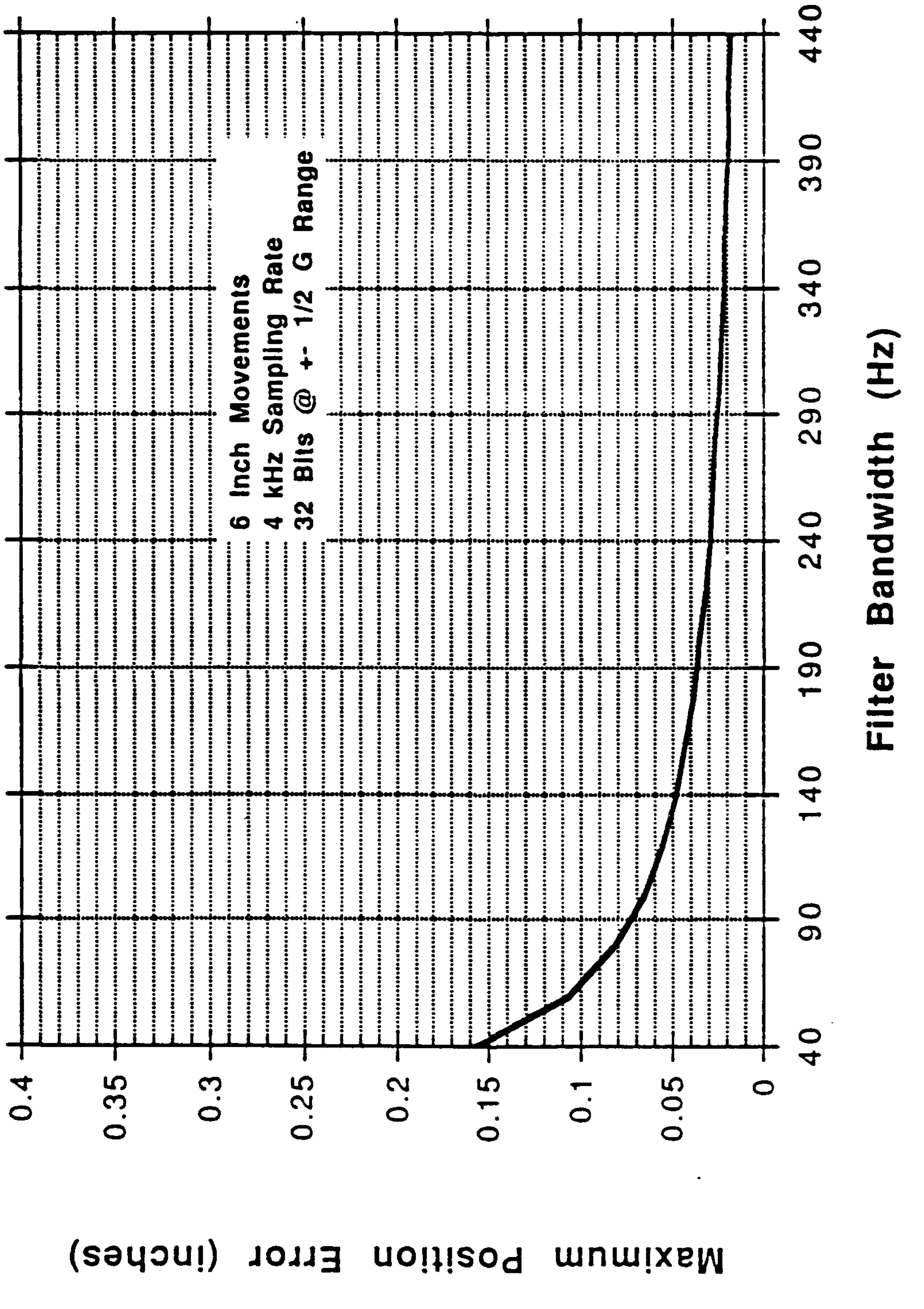


Figure 37 - Maximum Position Error versus Filter Bandwidth

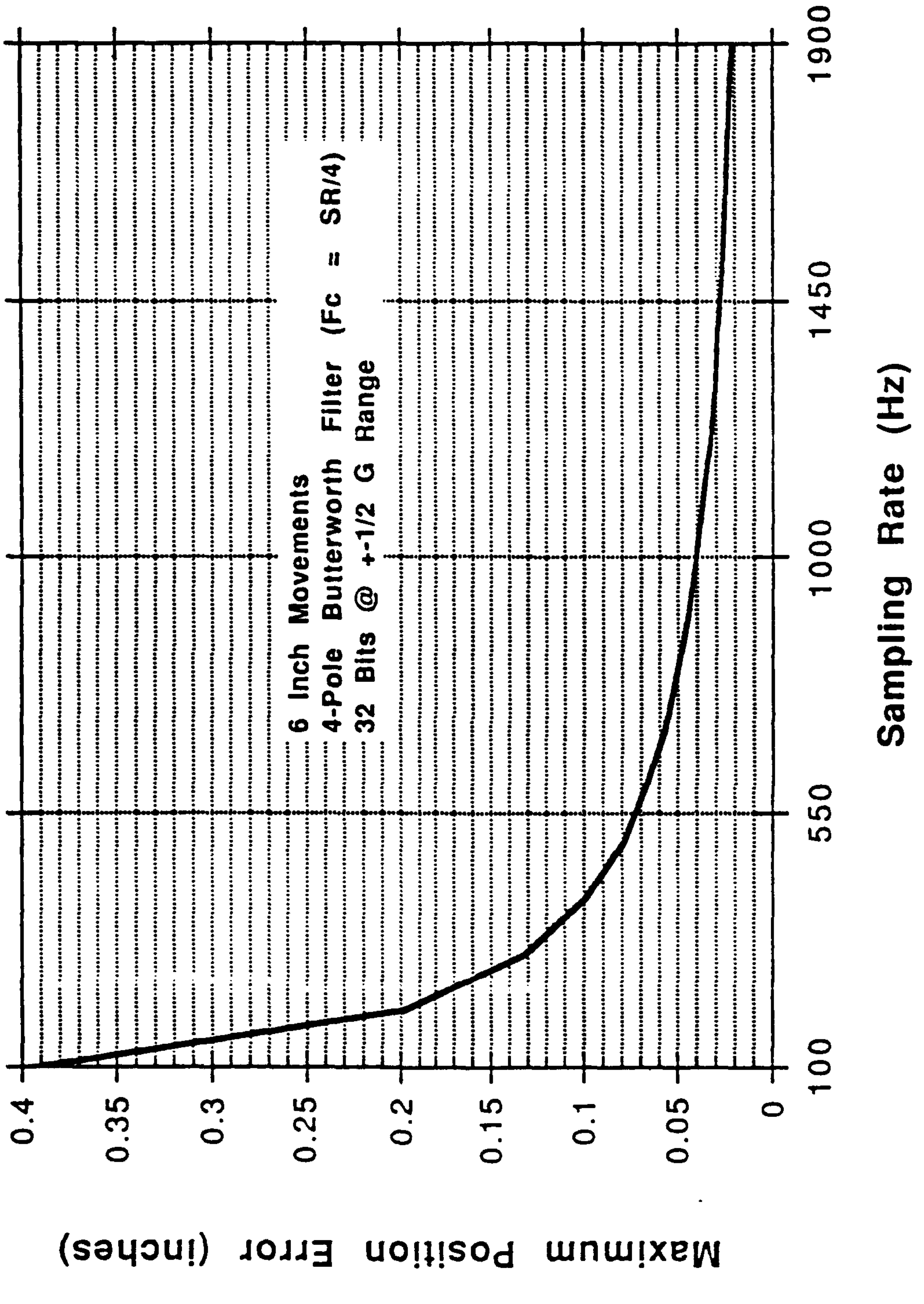


Figure 38 - Maximum Position Error versus DAS Sampling Rate

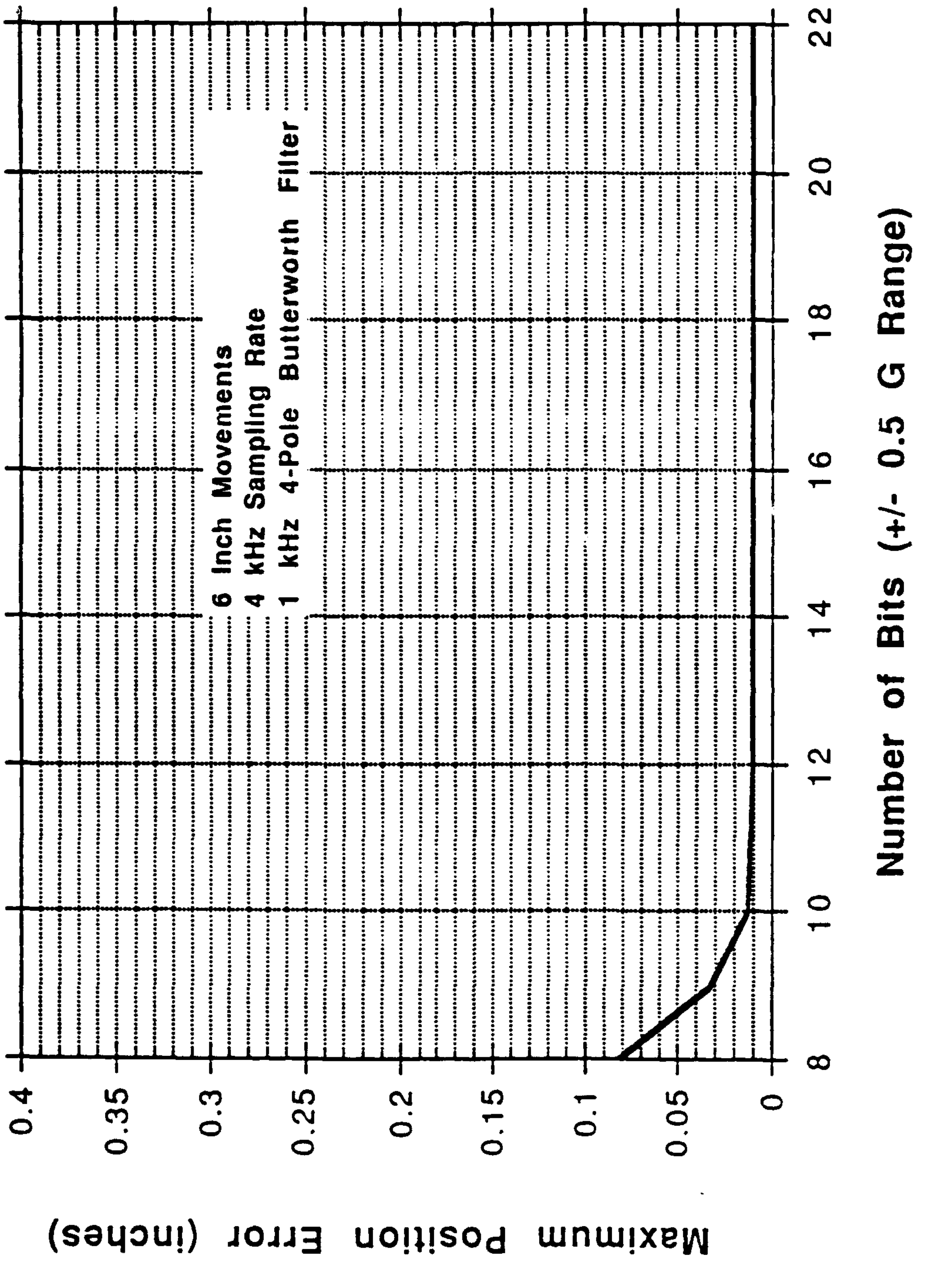


Figure 39 - Maximum Position Error versus DAS Resolution

5.2 Inertial System Parameters

The concept of IM based manipulator control is very attractive based on physical principles. However, the use of IMU has its difficulties because of sensor imperfections. Referring to Figure 4, one sees that the block representing IMU is in the feedback path. It is well known in feedback control theory that any uncertainly in a feedback element affects the system output directly and its effect cannot be lessened by the use of feedback. Therefore, finding ways to sufficiently reduce the uncertainties in an IMU is the key to a successful development of IM based control. Imperfections in an IMU mainly come from its inertial sensors, namely, accelerometers and gyros. Accelerometers suffer from bias uncertainties, gyros suffer from drift uncertainties, and they all suffer from scale factor uncertainties and nonlinearities. These uncertainties are slowly time varying quantities in general, which may become excessive over a sufficiently long period of time.

5.2.1 Inertial Bias Uncertainties

The precision of commercial accelerometers range from 10 micro-g to 10,000 micro-g, where g = 9.8 meters/sec, and that of commercial gyros range from .001 degree/hr to 100 degrees/hr. Consider an IMU equipped with high grade inertial sensors having the following uncertainties:

Accelerometer bias: 10 micro-g
Gyro drift: .01 degree/hr

Then, over an one minute period, the accumulated position error is about 18 centimeters, and the accumulated attitude error is about .6 arcsecond. The position error is not acceptable for most manipulator applications. If the accelerometer bias uncertainty can be reduced to 1 micro-g, then the position error will be 1.8 centimeters which is acceptable for some applications. The IMU imperfection problem may be solved by using a certain novel reinitialization technique during the course of manipulator operation. It is hoped that sensor uncertainties will change only slightly over a very short period, say, a few minutes. Then, at the end of each short period, the IMU is reinitialized to reduce the values of uncertainties. By so doing, the cumulative errors of the IMU can be kept sufficiently low. In the present study, a robot manipulator having only one-dimensional linear motion in the horizontal plane is considered. IM based control of such a system requires only a single accelerometer as inertial sensor.

5.3 Numerical Integration

A numerical integrator is needed to convert acceleration data to velocity and position data. Since a numerical integrator is an approximation to the ideal integrator, it causes errors. Three numerical integrators are compared. The three are trapezoidal rule integrator, Simpson's rule integrator, and Adam-Bashforth integrator. Their time domain algorithms and associated frequency domain transfer functions are given in the following:

Trapezoidal Rule Integrator

Algorithm:

$$y_k = y_{k-1} + (x_k + x_{k-1})*T/2 \qquad (1)$$

Transfer function:

$$H_T(z) = \frac{T\ (1 + z^{-1})}{2\ (1 - z^{-1})} \qquad (2)$$

Simpson's Rule Integrator

Algorithm:

$$y_k = y_{k-1} + (5x_k + 8x_{k-1} - x_{k-2})*T/12 \qquad (3)$$

Transfer function:

$$H_S(z) = \frac{T\ (1 + 4z^{-1} + z^{-2})}{3\ (1 - z^{-1})} \qquad (4)$$

Adam-Bashforth Integrator

Algorithm:

$$y_k = y_{k-1} + (55x_k - 59x_{k-1} + 37x_{k-2} - 9x_{k-3})*T/24 \qquad (5)$$

Transfer function:

$$H_A(z) = \frac{T\ (55 - 59z^{-1} + 37z^{-2} - 9z^{-3})}{24\ (1 - z^{-1})} \qquad (6)$$

The comparison is done by comparing the frequency responses of the three numerical integrators. The sampling frequency used in this test in 100 hertz. Figure 40 shows the magnitude and phase responses of the three numerical integrators and the ideal integrator. The gain of Simpson integrator becomes infinite at the folding frequency while the gain of trapezoidal integrator becomes zero at that frequency. These phenomena can be explained with the help of the pole-zero diagrams of integrator transfer functions as shown in Figure 41. Notice that the Simpson integrator has a pole at the folding frequency, which accounts for its infinite gain at that frequency. On the other hand, the trapezoidal integrator has a zero at the folding frequency making its gain zero there. All three integrators have pole at d-c.

To further compare these algorithms, the time-domain root-sum-square-error was calculated at different frequencies for a folding frequency of 50 Hz and tabulated in Table 9. This table shows Simpson's rule to be the best up to about 40 Hz, the Trapeziodal method is best from 40 to 45, and the Adam-Basforth algorithm is best from 45 to 50. Therefore, if the signal bandwidth is below an anti-aliasing filter with a bandwidth below $0.4f_S$, then the Simpson's Rule integrator is the best.

Table 9 - Time-domain Comparison of Three Numerical Integrators

Frequency (Hz)	Root-Sum-Square-Error		
	Adam-Basforth	Simpson	Trapeziodal
5	0.34199	0.00006	0.00321
10	0.12252	0.00047	0.00646
15	0.08159	0.00160	0.00976
20	0.08688	0.00382	0.01318
25	0.08266	0.00368	0.01673
30	0.06578	0.01379	0.02048
35	0.04446	0.02422	0.02449
40	0.02896	0.04495	0.02883
45	0.02848	0.10783	0.03362
50	0.04502	0.04502	0.04502

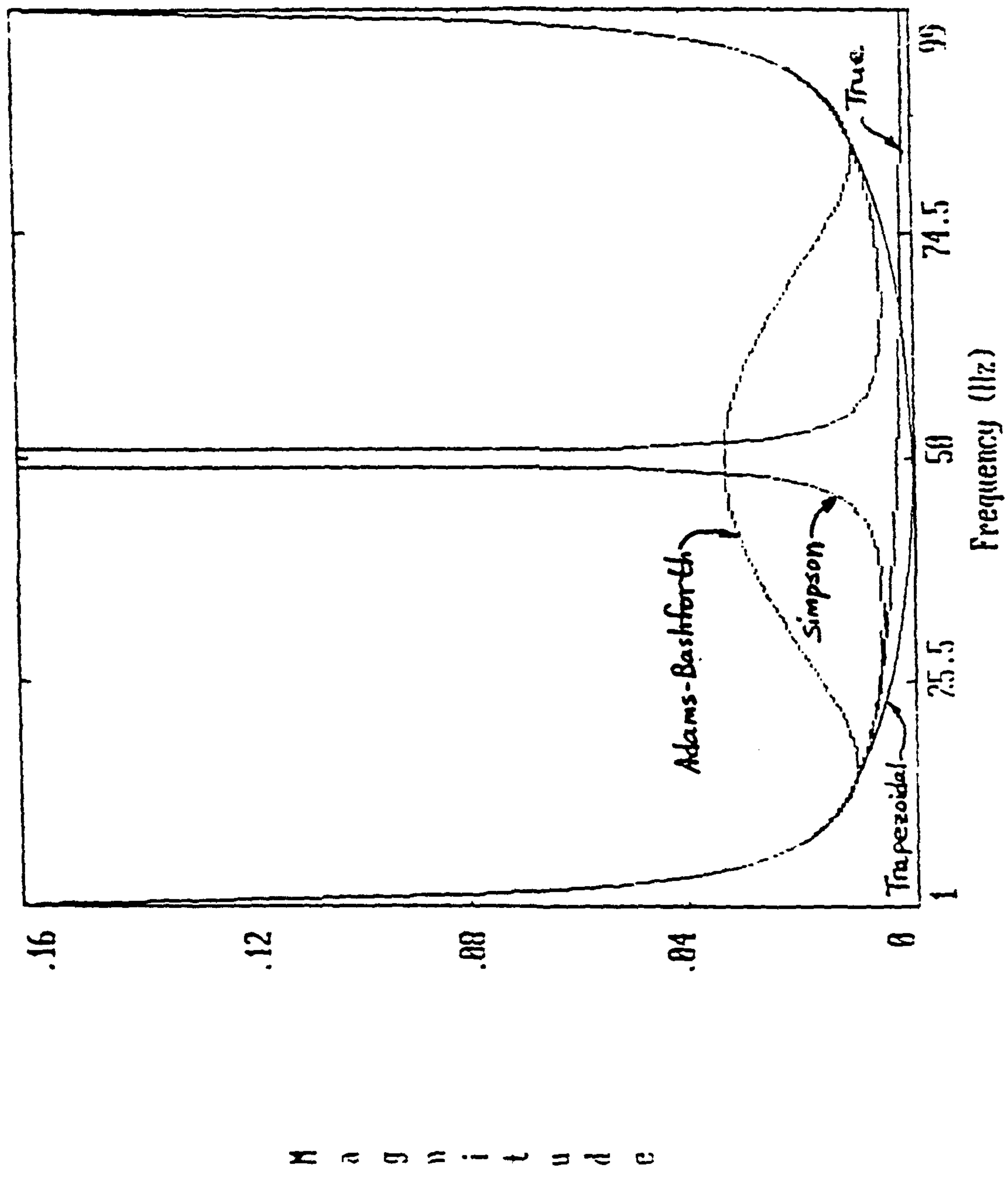
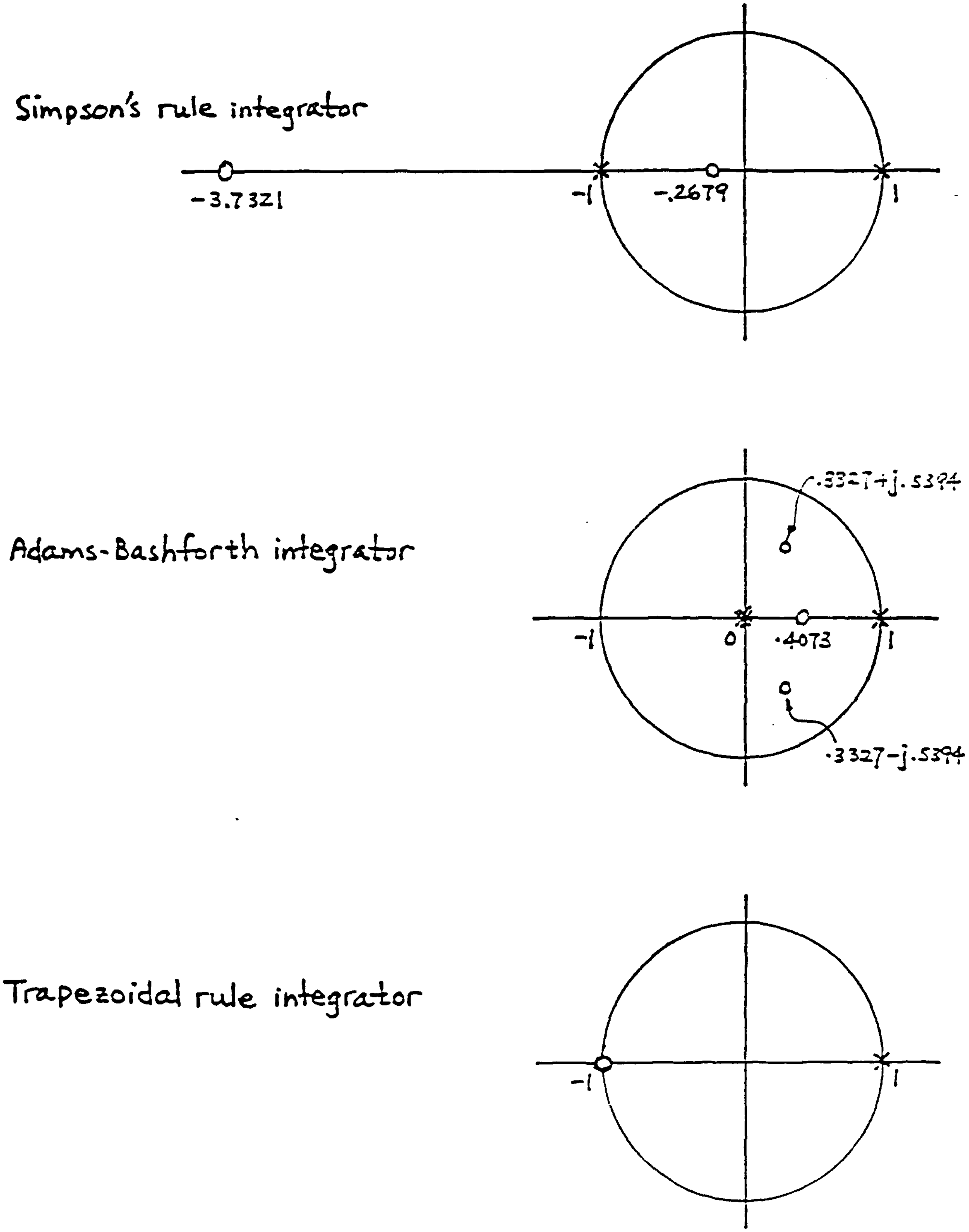


Figure 40 - Frequency Responses cf Three Numerical Integrators and the Ideal integrator



Simpson's rule integrator

Adams-Bashforth integrator

Trapezoidal rule integrator

Figure 41 - Pole-zero Diagrams of Three Numerical Integrators

6 Inertial Error Compensation Schemes

As discussed earlier, the growth of IMS errors can be contained by frequent reinitializations during the course of manipulator operation. Different initialization schemes, with different degrees of sophistication, can be devised. It is assumed that the accelerometer can be modeled by

$$m = Ka + B \qquad (9)$$

where m is the measured acceleration, a is the true acceleration, K is the scale factor, and B is the bias. The bias if further modeled as a linear function of time t over a short time period, that is,

$$B = B_0 + B_1 t \qquad (10)$$

where B_0 and B_1 are constants. The purpose of frequent reinitialization is to determine the parameters of the model and make corrections for velocity and position periodically (~once a minute). In the following, three reinitialization schemes, with increasing degree of sophistication, are presented.

6.1 Zero-Velocity Update

Zero-velocity update, abbreviated ZUPT by the navigation profession, is the *simplest initiation scheme.* It provides information to update only one accelerometer error parameter, usually the constant term B_0 of the bias. When end-effector stops, its true velocity is zero. Any nonzero velocity computed from accelerometer output is the velocity error V_{err} of the inertial measurement system. Assume uncertainties in K and B are negligible, one has the relationship

$$V_{err} = B_0 T \qquad (11)$$

where T is the time of elapse from the previous initialization. Thus B_o can be computed from

$$B_0 = V_{err}/T \qquad (12)$$

With B_0 known, the present end-effector position can be corrected using

$$S_{new} = S_{old} - B_0 t^2/4 \qquad (13)$$

where S_{old} and S_{new} are end-effector positions before and after the correction, respectively. The ZUPT software based on this principle is given in the form of a computer flow chart shown in Figure 42. Details in flow chart blocks are given below.

Given: One base-station, the home station
N, the number of motions
D_K, direction of the k-th motion, k=1 to N
D_K, distance for the k-th motion, k=1 to N
B, the initial accelerometer bias

K_{2F}, the accelerometer scale factor

Block 1. Read initial reference position $x_{ref}(0)$.
Set initial IAR distance D (0)=0.
Set k=1, the first motion.
Set the total distance traveled D_T=0.
Set the total computation time steps i_T=0.
Set the present computation time step i=1.

Block 2. Command motion to start.

Block 3. Read accelerometer output data a(i).

Block 4. Integrate a(i) to give velocity v(i).

Block 5. Is D(i) = D_K?

Block 6. Command motion to stop.

Block 7. Compute accelerometer bias uncertainty

$$T = i \text{ x at}$$
$$B_0 = \frac{V(i)}{T}$$

where t is the sampling period. Update bias by

$$B = B + B_0$$

Compute position correction

$$x = x - ¼ B_0T^2$$

Compute the total time steps $i_T = i_T + i$.
Compute the total distance $D_T = D_T + D(i)$.
Reset i=0.

Block 8. Read reference position $x_{ref}(i_T)$.

Block 9. Compute reference distance
$D_{ref} = x_{ref}(i_T) - x_{ref}(0)$.

Compute error in IAR distance $e = D(i_T) - D_{ref}$.

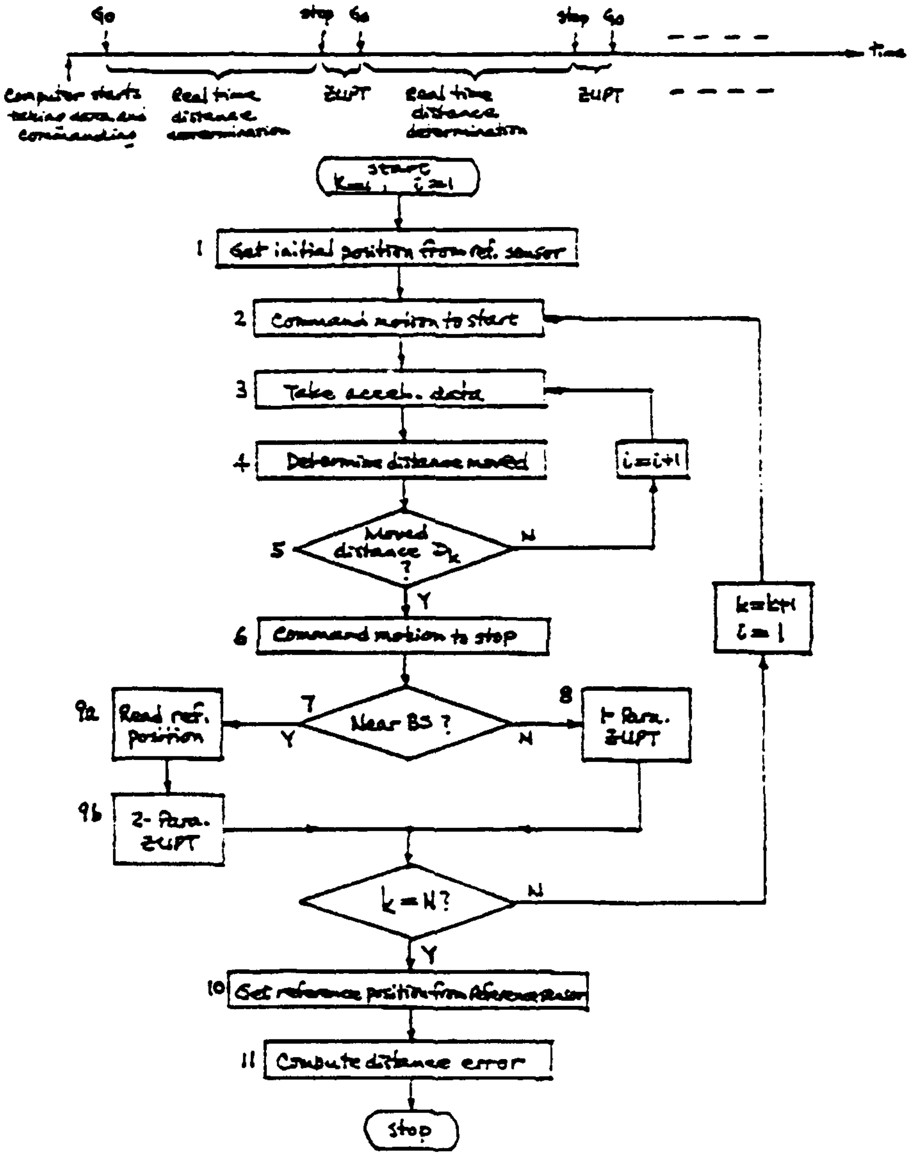

Go
stop Go
stop Go
– – – –
time
Computer starts taking data and commanding
Real time distance determination
ZUPT
Real time distance determination
ZUPT
– – – –
start
K=1 , i=1
1 Get initial position from ref. sensor
2 Command motion to start
3 Take accel. data
4 Determine distance moved
i=i+1
5 Moved distance D_k ?
N
Y
6 Command motion to stop
k=k+1
i=1
7 Near BS ?
Y
N
9a Read ref. position
9b 2- Para. ZUPT
8 1- Para. ZUPT
k=N?
N
Y
10 Get reference position from Reference sensor
11 Compute distance error
stop

Figure 42 - Block Diagram for Scheme 1

6.2 Zero-Velocity Update with Round-Trip Motion

If the motion of the end-effector consists of one or more round-trip stops, then at each round-trip stop the true velocity is zero and the true net distance is also zero. Any non-zero values of velocity and distance, computed using accelerometer output, are errors. Let V_{err} and P_{err} be the non-zero velocity and distance values at the end of a round-trip, and let T be the total time for the round-trip. The V_{err} and P_{err} are related to the bias coefficient uncertainties B_0 and B_1 through the following equation.

$$V_{err} = TB_0 + \frac{T^2}{2}B_1$$
$$P_{err} = \frac{T^2}{2}B_0 + \frac{T^3}{6}B_1 \qquad (14)$$

Solving the above equations for B_0 and B_1, gives

$$B_0 = \frac{2}{T} V_{err} + \frac{6}{T^2}P_{err}$$
$$B_1 = \frac{6}{T^2} V_{err} - \frac{12}{T^3} P_{err} \qquad (15)$$

Therefore two accelerometer bias coefficients can be updated. Figure 43 is a computer software flow chart for the scheme of zero-velocity update with round-trip motion. Details in flow chart boxes are given below.

Given: One base-station, the home station
N, the total number of motions
d_K, the direction of the k-th motion, k=1 to N.
D_K, the distance of the k-th motion, k-1 to N.
Accelerometer bias $B = B_0 + B_1t$.
Accelerometer scale factor K_{st}.
Stops which are round-trip stops.

Block 1. Read initial reference position $x_{ref}(0)$.
Set initial distance D(0)=0.
Set motion number k=1.
Set the total distance traveled D_T=0.
Set the time step count, for each motion, i-i.
Set the total time steps i_T= 0.

Block 2. Command motion to start.

Block 3. Read accelerometer output data a(i).

Block 4. Integrate a(i) to give v(i).
integrate v(i) to give D(i).

Block 5. Is $D(i)=D_K$?

Block 6. Command motion to stop.

Block 7. Is this end of a round-trip?

$$T = ix \quad t$$

$$B_0 = \frac{V(i)}{T}$$

Update bias $B_0 = B_0 + \ B_0$.
Compute position correction

$$x = x - \frac{1}{2} B_0 T^2$$

Compute total time steps $i_T = i_T + i$.
Compute total distance $D_T = D_T + D(i)$.

Block 9a. Read reference position $x_{ref}(i)$.

Block 9b. Compute IAR distance error

$$D = D(i) - [x_{ref}(i) - x_{ref}(0)]$$

Compute accelerometer bias coefficients

$$B_0 = -\frac{2}{T} V(i) + \frac{6}{T^2} \ D$$

$$B_1 = -\frac{6}{T^2} V(i) + \frac{12}{T^3} \ D$$

Compute

$$\begin{aligned} B_0 &= B_0 - \ B_0 \\ B_1 &= B_1 - \ B_1 \\ i_T &= i_T + i \\ D_T &= D_T + D(i) \end{aligned}$$

Compute position correction

$$x = x_{ref}$$

Reset i=i.

Block 10. Read reference position $x_{ref}(i_T)$.

Block 11. Compute reference distance

$$D_{ref} = X_{ref}(i) - X_{ref}(0)$$

Compute error in IAR distance

$$E = D(i_T) - D_{ref}$$

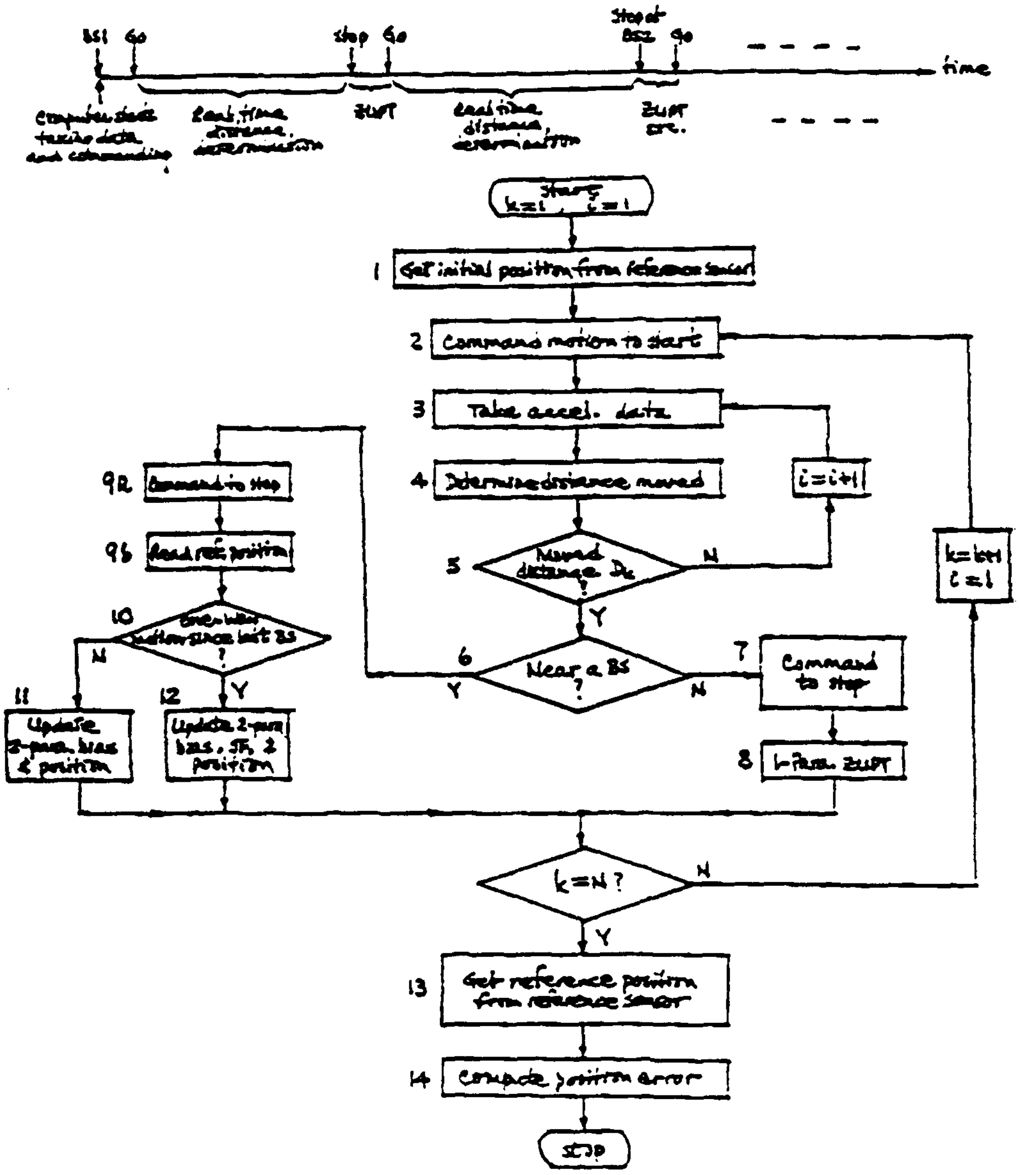

BS1
Go
Stop
Go
Stop at BS2
Go
time
Computer slack taking data and commanding
Real time distance determination
ZUPT
Real time distance determination
ZUPT etc.
Start k=1, i=1
1 Get initial position from reference sensor
2 Command motion to start
3 Take accel. data
4 Determine distance moved
i=i+1
5 Moved distance Dc ?
N
Y
6 Near a BS ?
Y
N
7 Command to stop
8 1-Para. ZUPT
9a Command to stop
9b Read ref. position
10 One-way motion since last BS ?
N
Y
11 Update 2-para. bias & position
12 Update 2-para bias, SF, & position
k=k+1
i=1
k=N ?
N
Y
13 Get reference position from reference sensor
14 Compute position error
STOP

Figure 43 - Block Diagram for Scheme 2

6.3 Zero-Velocity Update and Two Reference Stations

This scheme posseses all the features of previous schemes with the addition that two base-stations are available, the scale factor of the accelerometer can be updated if the motion from one base-station to the other is in all in one direction. The distance traveled is

$$D_{TR} = X_{RS2} - X_{RS1} \qquad (16)$$

where X_{RS1} and X_{RS2} are positions of the first and second base-stations, respectively, and D_{TR} is the true distance between the two base-stations. Then the scale factor correction factor is given by

$$\beta_8 = D_{TR}/D_{comp} \qquad (17)$$

The updated scale factor is

$$K_{new} = \beta_8 K_{old} \qquad (18)$$

The above scale factor update procedure is used in the real-time menu option labeled with (M1). The menu option labeled (M2) updates the scale factor by averaging the above update with the last update,

$$K_{new} = (\beta_8 K_{old} + K_{old})/2 \qquad (19)$$

Figure 44 is the computer software flow chart for this scheme. Details in flow chart blocks are given below.

Given: Two base-stations, with positions $X_{RS2} - X_{RS1}$
N, the total number of motions
d_K, the direction of the k-th motion, k=1 to N.
D_K, the distance of the k-th motion, k-1 to N.
which stops are at base-station
Accelerometer bias $B = B_0 + B_1 t$.
Accelerometer scale factor K_{st}.

Block 1. Read initial reference position $x_{ref}(0)$.
Set initial IAR distance D(0)=0.
Set initial position $x(0) = x_{ref}(0)$.
Set motion number k=1.
Set the time step count, for each motion, i-i.
Set the total distance traveled D_T=0.
Set the total time steps i_T= 0.

Block 2. Command motion to start.

Block 3. Read accelerometer output data a(i).

Block 4. Integrate a(i) to give a(i).
Integrate v(i) to give D(i).

Block 5. Is $D(i) = D_K$?

Block 6. Is this a base-station stop?

Block 7. Command motion to stop.

Block 8. Compute bias uncertainty

$$T = ix \quad t$$

$$B_0 = \frac{V(i)}{T}$$

Compute updates

$$B_0 = B_0 \;+\; B_0$$

$$i_T = i_T + i$$

$$D_T = D_T + D(i)$$

Reset i=0.
Compute position correction

$$x = x - \frac{1}{4} \; B_0 T^2$$

Block 9a. Command motion to stop.

Block 9b. Read reference position $x_{ref}(i)$.

Block 10. Is this a one way motion from last base-station?

Block 11. Compute IAR distance error

$$D = D(i) - [x_{ref}(i) - x_{ref}(0)]$$

Compute bias coefficients

$$B_0 = \frac{2}{T} V(i) + \frac{6}{T^2} D$$

$$B_1 = \frac{6}{T^2} V(i) + \frac{12}{T^3} D$$

$$B_0 = B_0 + \; B_0$$

$$B_1 = B_1 + \; B_1$$

Set $i_T = i_T + i$ and $D_T = D_T + D(i)$.

Reset IAR position $x(0)=x_{ref}(i)$.

Reset initial reference position $x_{ref}(0)=x_{ref}(i)$.

Reset i = 0.

Block 12. Same as Block 11 with additional scale factor update computations.

$$B_{SF} = \frac{\text{Actual distance between 2 base-stations}}{\text{Computed distance between 2 base-stations}}$$

$$K_{SF} = B_{SF}K_{SF}$$

Block 13. Read reference position x (i).

Block 14. Compute error in IAR position

$$E = x(i) - X_{ref}(i_T)$$

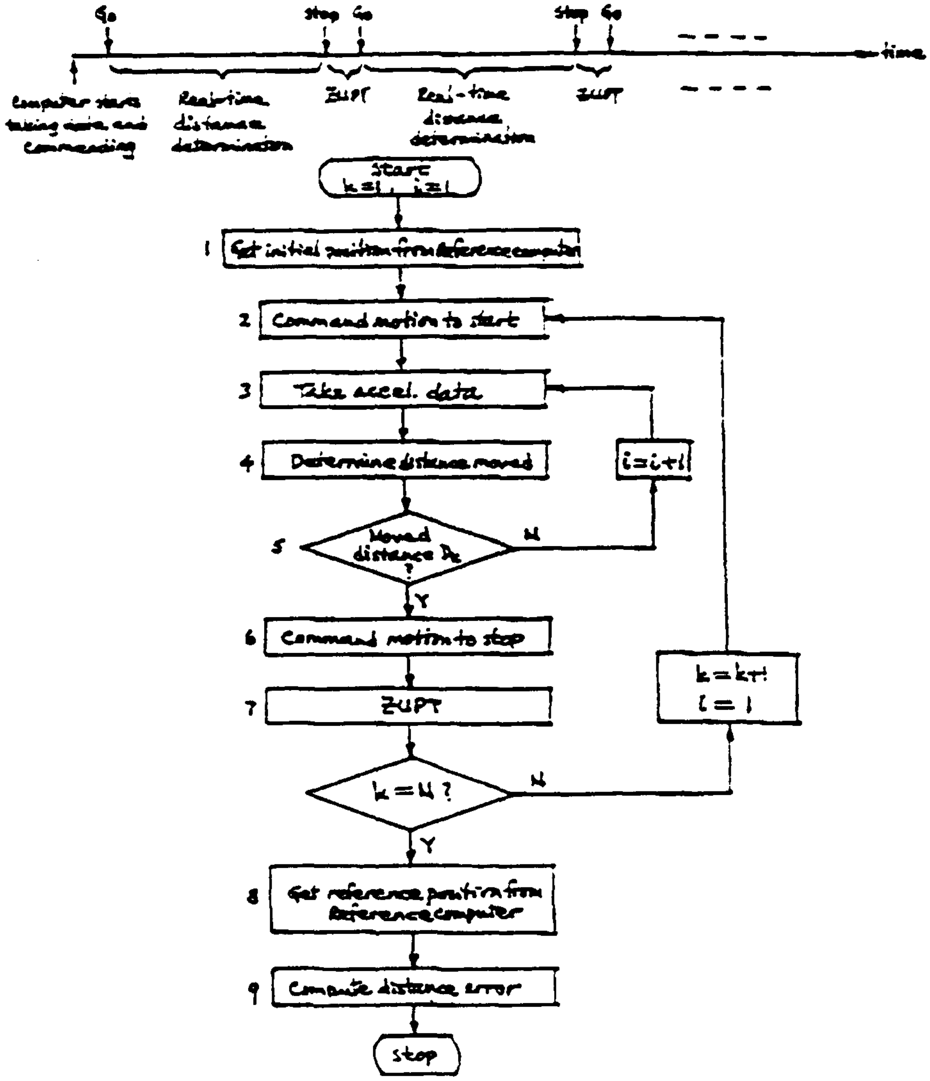


Figure 44 - Block Diagram for Scheme 3

7 Real-time Position Determination Software

The real-time position determination program is a menu driven environment that allows the user to setup and test a single-axis inertial positioning system. The program was written in the "c" programming language and compiled using the Microsoft C Version 5.1 Optimizing Compiler. The source code is included in Appendix C. The modular design of the program will allow a different DAS, translation table, or processing unit to be used by only modifying a few subroutines. A flow chart of the program is given in Figure 45.

The main menu offers four options:

1) **System Setup** - Activates a sub-menu with choices to calibrate the AD1170's, calibrate the QA2000 scale factor and bias, select one of three integration methods, or change the DAS sampling frequency.

2) **Experiments** - Activates a sub-menu with that allows the user to select from five application scenario tests. A log file will record each test that is run along with the results.

3) **Free-Running Position Display** - Uses the currently selected sampling rate, and integration algorithm along with the last updates of the scale factor and bias to display the real-time position of the accelerometer. The program will capture any key-strokes and send them to the Anorad table so that the accelerometer can be moved. (Appendix D contains a brief description of all of the Anorad table commands.) A no-motion command signal can be generated by pressing "Z" which will invoke a zero velocity update. To return to the main menu press "*".

4) **Quit** - Quits the program.

The five application scenarios are actually tests of the compensation schemes discussed in section 6. The tests are referred to in the program menu as:

1) **Single Motion Test** - Does not use any compensation. One run equals one 3.6 inch move.

2) **Multiple Motion Test w/ZUPT** - Updates the accelerometer bias by simply averaging the output of the accelerometer inbetween movements. One run equals three 3.6 inch moves. Discussed in section 6.1.

3) **Multiple Motion Test w/ZUPT & Unknown Base** - Updates the accelerometer bias by using the position error and velocity error over one run. One run equals two 3.6 inch moves - one forward and one backward. Discussed in section 6.2

4) **Multiple Motion Test w/ZUPT & Known Base** - Updates the

4) **Multiple Motion Test w/ZUPT & Known Base** - Updates the accelerometer bias by using the position error and velocity error over one run. One run equals two 3.6 inch moves - one forward and one backward. Discussed in section 6.2

5) **Multiple Motion Test w/ZUPT & 2 Known Base (M1)** - Updates the accelerometer bias and scale factor by using the position error and velocity error over one run. One run equals two 3.6 inch moves in the same direction. Discussed in section 6.3

6) **Multiple Motion Test w/ZUPT & 2 Known Base (M2)** - Updates the accelerometer bias by using a regular ZUPT and updates the scale factor using the position error. One run equals two 3.6 inch moves in the same direction. Discussed in section 6.3

The results of each test run along with the statistics and experimental conditions for each set or runs are stored in a log file. A plot of the acceleration profile for the 3.6 inch movement used in most of these tests is shown in Figure 46.

The temperature compensation was implemented implicitly since the bias and scale factor (SF) updates would take into account the variations due to temperature changes. This implicit method is better since the temperature coeffiecients vary significantly from one measurment to another as determined in phase one of this project.

The heart of the real-time software is a subroutine called "integrate_and_move" which simultaneously determines the real-time position and controls the Anorad table. The inputs to the routine are amount of time to determine position in seconds and a character string of Anorad table commands, while the output is the distance moved in inches after the specified time.

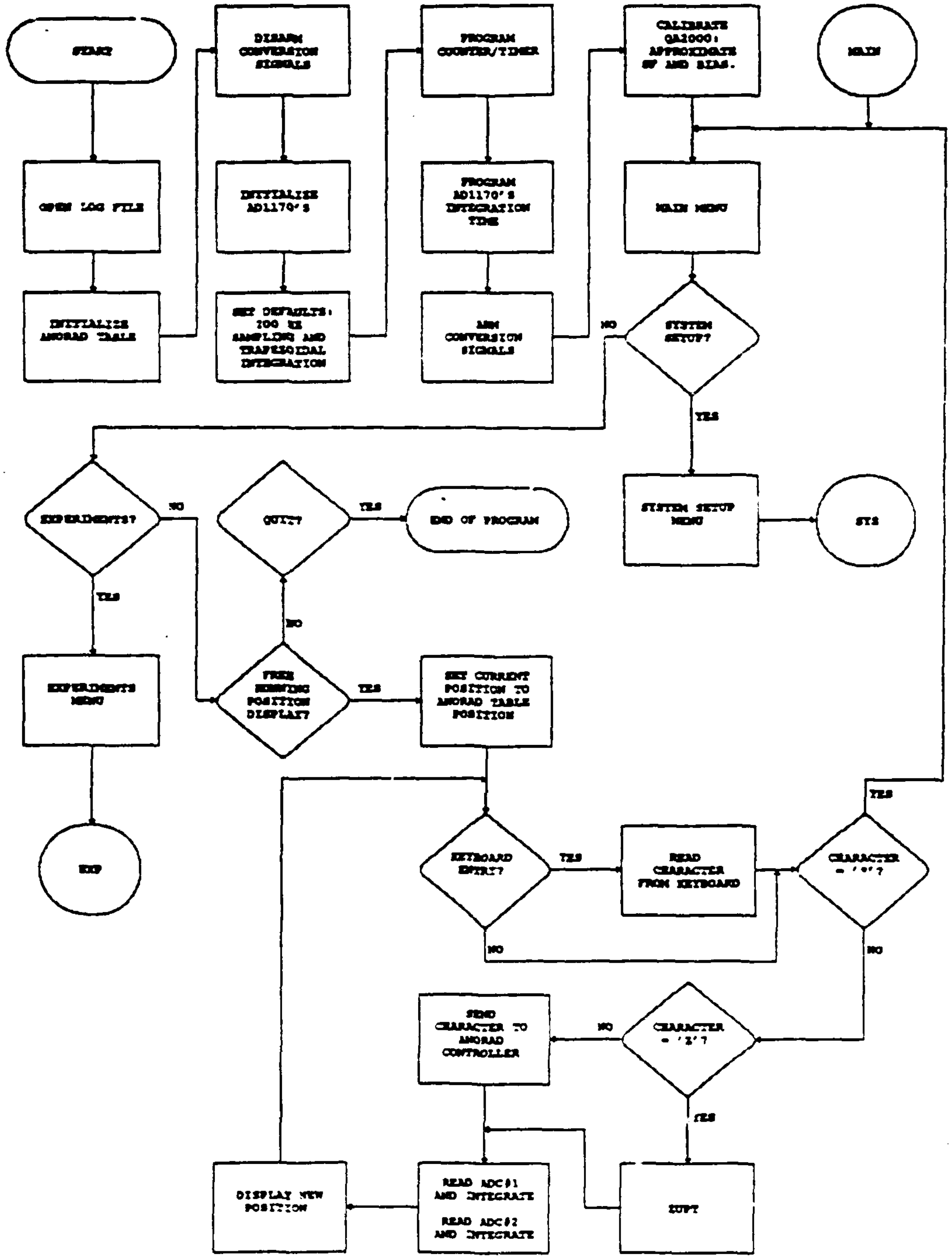


Figure 45 - Real-time Position Determination Program Flow Chart

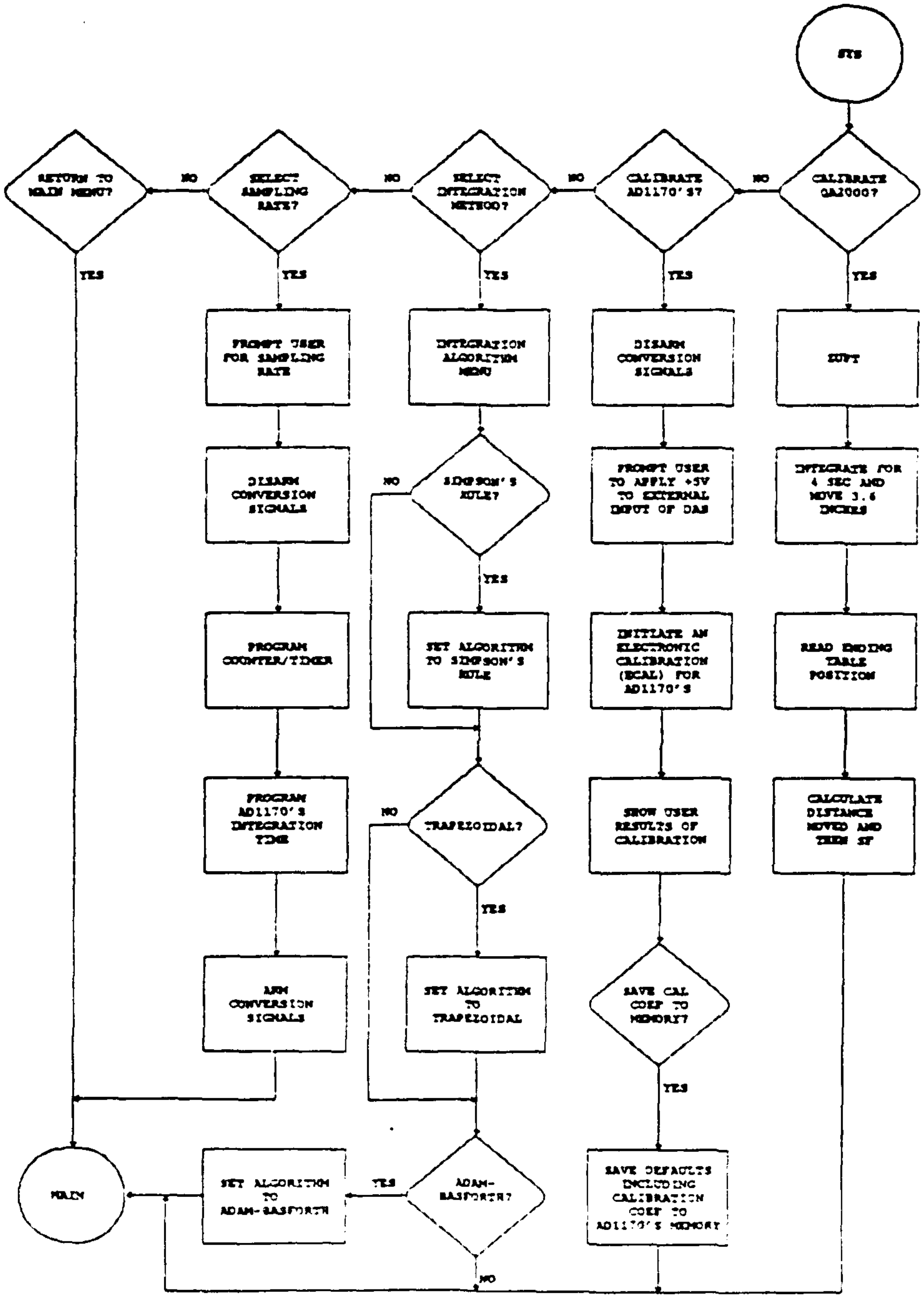

SYS
RETURN TO MAIN MENU?
NO
SELECT SAMPLING RATE?
NO
SELECT INTEGRATION METHOD?
NO
CALIBRATE AD1170'S?
NO
CALIBRATE QA2000?
YES
YES
YES
YES
YES
PROMPT USER FOR SAMPLING RATE
INTEGRATION ALGORITHM MENU
DISARM CONVERSION SIGNALS
ZUPT
DISARM CONVERSION SIGNALS
NO
SIMPSON'S RULE?
PROMPT USER TO APPLY +5V TO EXTERNAL INPUT OF DAS
INTEGRATE FOR 4 SEC AND MOVE 3.6 INCHES
YES
PROGRAM COUNTER/TIMER
SET ALGORITHM TO SIMPSON'S RULE
INITIATE AN ELECTRONIC CALIBRATION (ECAL) FOR AD1170'S
READ ENDING TABLE POSITION
PROGRAM AD1170'S INTEGRATION TIME
NO
TRAPEZOIDAL?
SHOW USER RESULTS OF CALIBRATION
CALCULATE DISTANCE MOVED AND THEN SF
YES
ARM CONVERSION SIGNALS
SET ALGORITHM TO TRAPEZOIDAL
SAVE CAL COEF TO MEMORY?
YES
MAIN
SET ALGORITHM TO ADAM-BASFORTH
YES
ADAM-BASFORTH?
SAVE DEFAULTS INCLUDING CALIBRATION COEF TO AD1170'S MEMORY
NO

Figure 45 (Cont.) - Real-time Position Determination Program Flow Chart

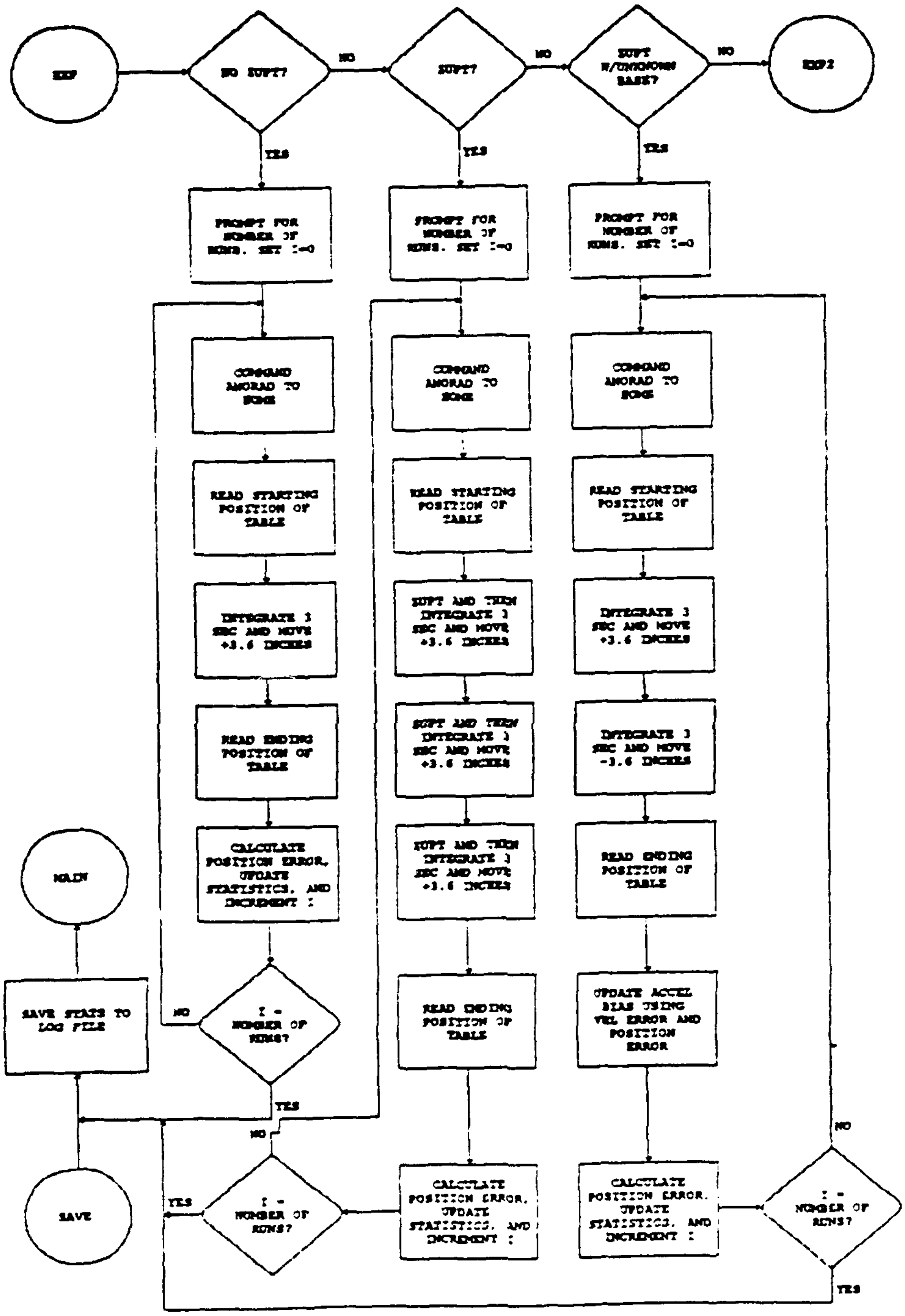

EXP
NO SUPT?
NO
SUPT?
NO
SUPT W/UNKNOWN BASE?
NO
EXP2
YES
YES
YES
PROMPT FOR NUMBER OF RUNS. SET I=0
PROMPT FOR NUMBER OF RUNS. SET I=0
PROMPT FOR NUMBER OF RUNS. SET I=0
COMMAND AMORAD TO HOME
COMMAND AMORAD TO HOME
COMMAND AMORAD TO HOME
READ STARTING POSITION OF TABLE
READ STARTING POSITION OF TABLE
READ STARTING POSITION OF TABLE
INTEGRATE 3 SEC AND MOVE +3.6 INCHES
SUPT AND THEN INTEGRATE 3 SEC AND MOVE +3.6 INCHES
INTEGRATE 3 SEC AND MOVE +3.6 INCHES
READ ENDING POSITION OF TABLE
SUPT AND THEN INTEGRATE 3 SEC AND MOVE +3.6 INCHES
INTEGRATE 3 SEC AND MOVE -3.6 INCHES
MAIN
CALCULATE POSITION ERROR, UPDATE STATISTICS, AND INCREMENT I
SUPT AND THEN INTEGRATE 3 SEC AND MOVE +3.6 INCHES
READ ENDING POSITION OF TABLE
SAVE STATS TO LOG FILE
NO
I = NUMBER OF RUNS?
READ ENDING POSITION OF TABLE
UPDATE ACCEL BIAS USING VEL ERROR AND POSITION ERROR
YES
NO
NO
SAVE
YES
I = NUMBER OF RUNS?
CALCULATE POSITION ERROR, UPDATE STATISTICS, AND INCREMENT I
CALCULATE POSITION ERROR, UPDATE STATISTICS, AND INCREMENT I
I = NUMBER OF RUNS?
YES

Figure 45 (Cont.) - Real-time Position Determination Program Flow Chart

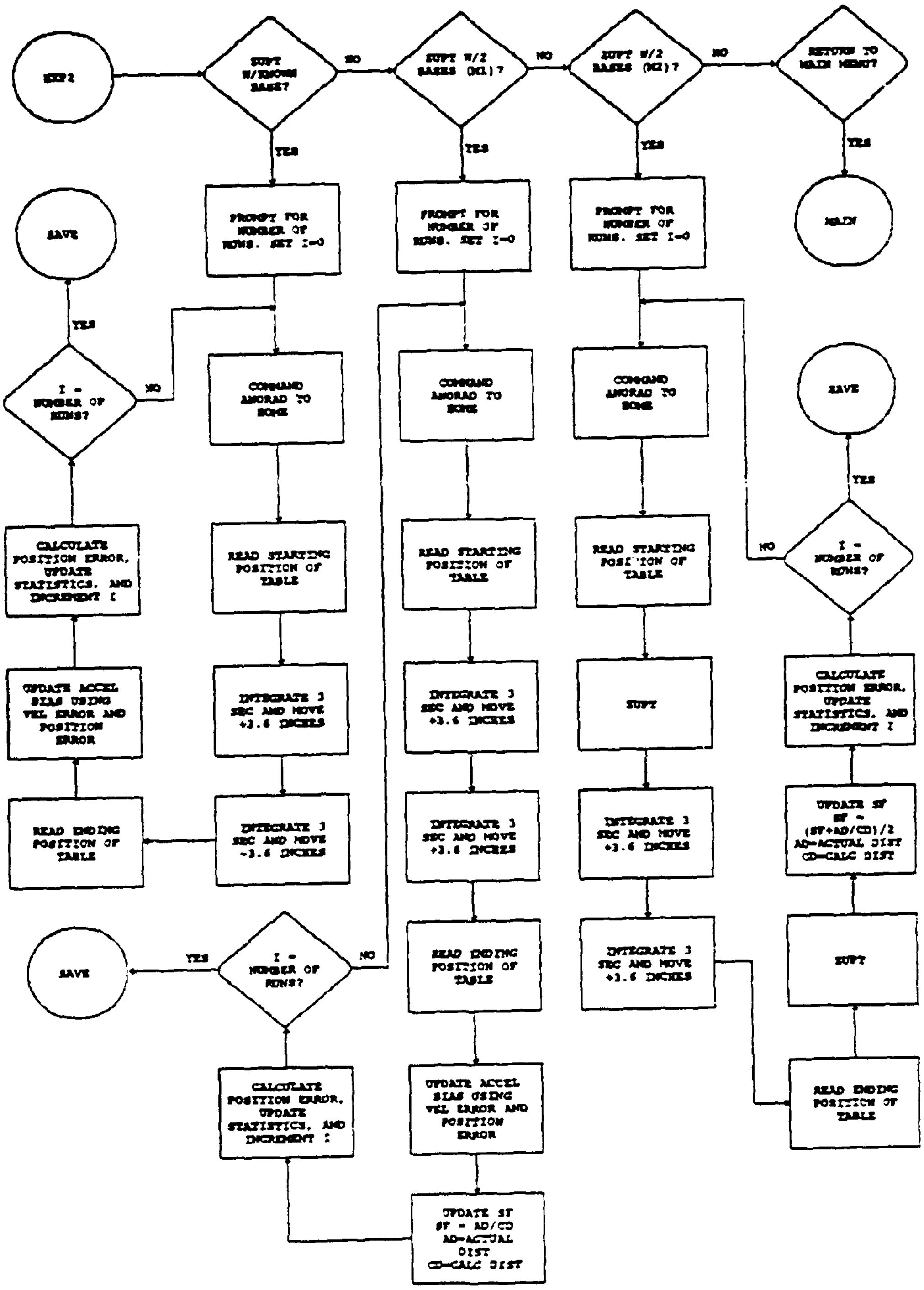

EXP2
SUPT W/KNOWN BASE?
NO
SUPT W/2 BASES (M1)?
NO
SUPT W/2 BASES (M2)?
NO
RETURN TO MAIN MENU?
YES
YES
YES
YES
SAVE
PROMPT FOR NUMBER OF RUNS. SET I=0
PROMPT FOR NUMBER OF RUNS. SET I=0
PROMPT FOR NUMBER OF RUNS. SET I=0
MAIN
YES
I = NUMBER OF RUNS?
NO
COMMAND AMORAD TO HOME
COMMAND AMORAD TO HOME
COMMAND AMORAD TO HOME
SAVE
YES
CALCULATE POSITION ERROR, UPDATE STATISTICS, AND INCREMENT I
READ STARTING POSITION OF TABLE
READ STARTING POSITION OF TABLE
READ STARTING POSITION OF TABLE
NO
I = NUMBER OF RUNS?
UPDATE ACCEL BIAS USING VEL ERROR AND POSITION ERROR
INTEGRATE 3 SEC AND MOVE +3.6 INCHES
INTEGRATE 3 SEC AND MOVE +3.6 INCHES
SUPT
CALCULATE POSITION ERROR, UPDATE STATISTICS, AND INCREMENT I
READ ENDING POSITION OF TABLE
INTEGRATE 3 SEC AND MOVE -3.6 INCHES
INTEGRATE 3 SEC AND MOVE +3.6 INCHES
INTEGRATE 3 SEC AND MOVE +3.6 INCHES
UPDATE SF SF = (SF+AD/CD)/2 AD=ACTUAL DIST CD=CALC DIST
SAVE
YES
I = NUMBER OF RUNS?
NO
READ ENDING POSITION OF TABLE
INTEGRATE 3 SEC AND MOVE +3.6 INCHES
SUPT
CALCULATE POSITION ERROR, UPDATE STATISTICS, AND INCREMENT I
UPDATE ACCEL BIAS USING VEL ERROR AND POSITION ERROR
READ ENDING POSITION OF TABLE
UPDATE SF SF = AD/CD AD=ACTUAL DIST CD=CALC DIST

Figure 45 (Cont.) - Real-time Position Determination Program Flow Chart

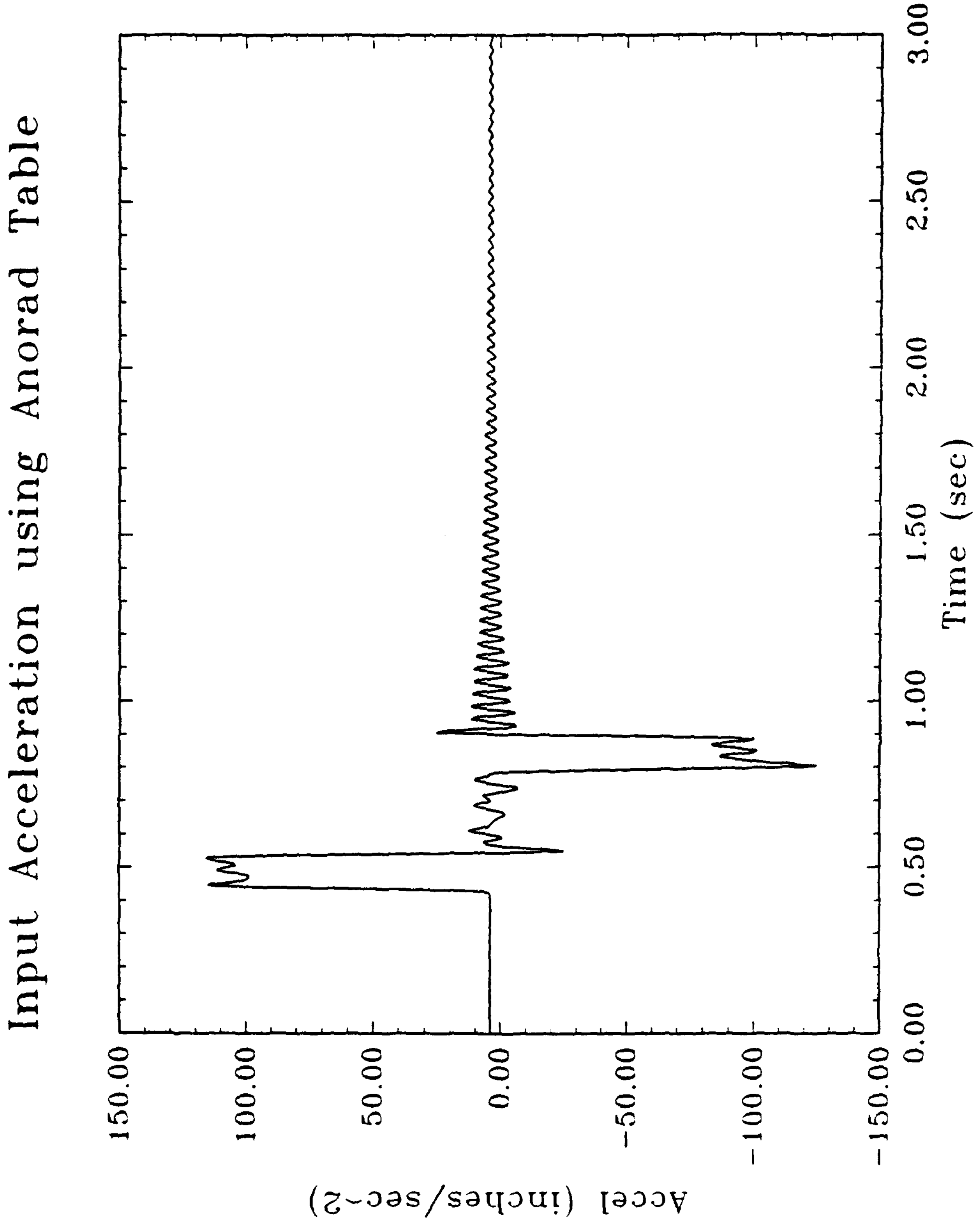


Figure 46 - Acceleration Profile of 3.6 Inch Move

8 System Tests

8.1 System Setup

The robot end-effector having one-dimensional linear motion was implemented using a linear motion table manufactured by Anorad Corporation, Hauppange, New York. The table was equipped with a motion controller contained in a separate box. Mounted on the table is an optical linear encoder which can measure the position of the table to 16 microinches. The table is maintained level. An accelerometer is mounted on the table with its input axis pointed along the direction of table motion. The output of the accelerometer goes to the DAS which contains interface electronics, anti-aliasing filter, and A/D converter. The output of A/D converter goes to a microcomputer (Compaq 386/25) which is also connected to the table controller via an RS232 port. The RS232 cable connections are shown in Figure 47. Figure 48 is a sketch of the system setup.

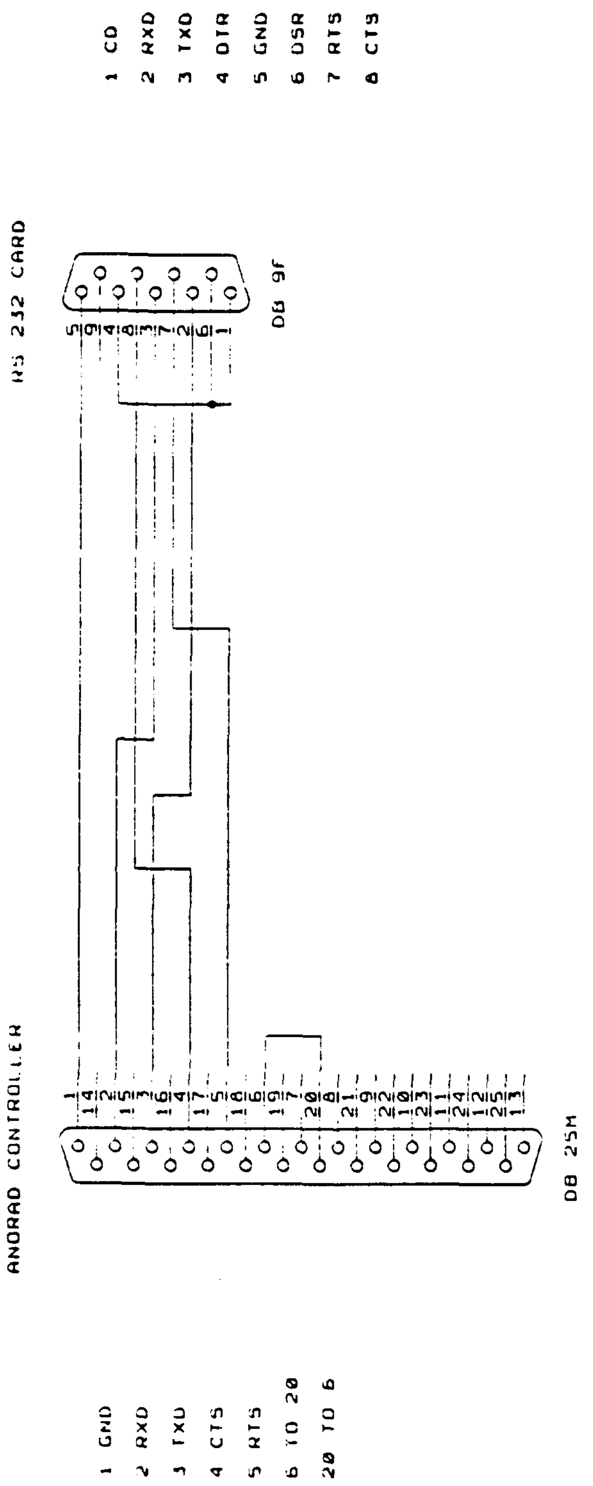


Figure 47 - Cable Connections for RS232 Communications with Anorad Table

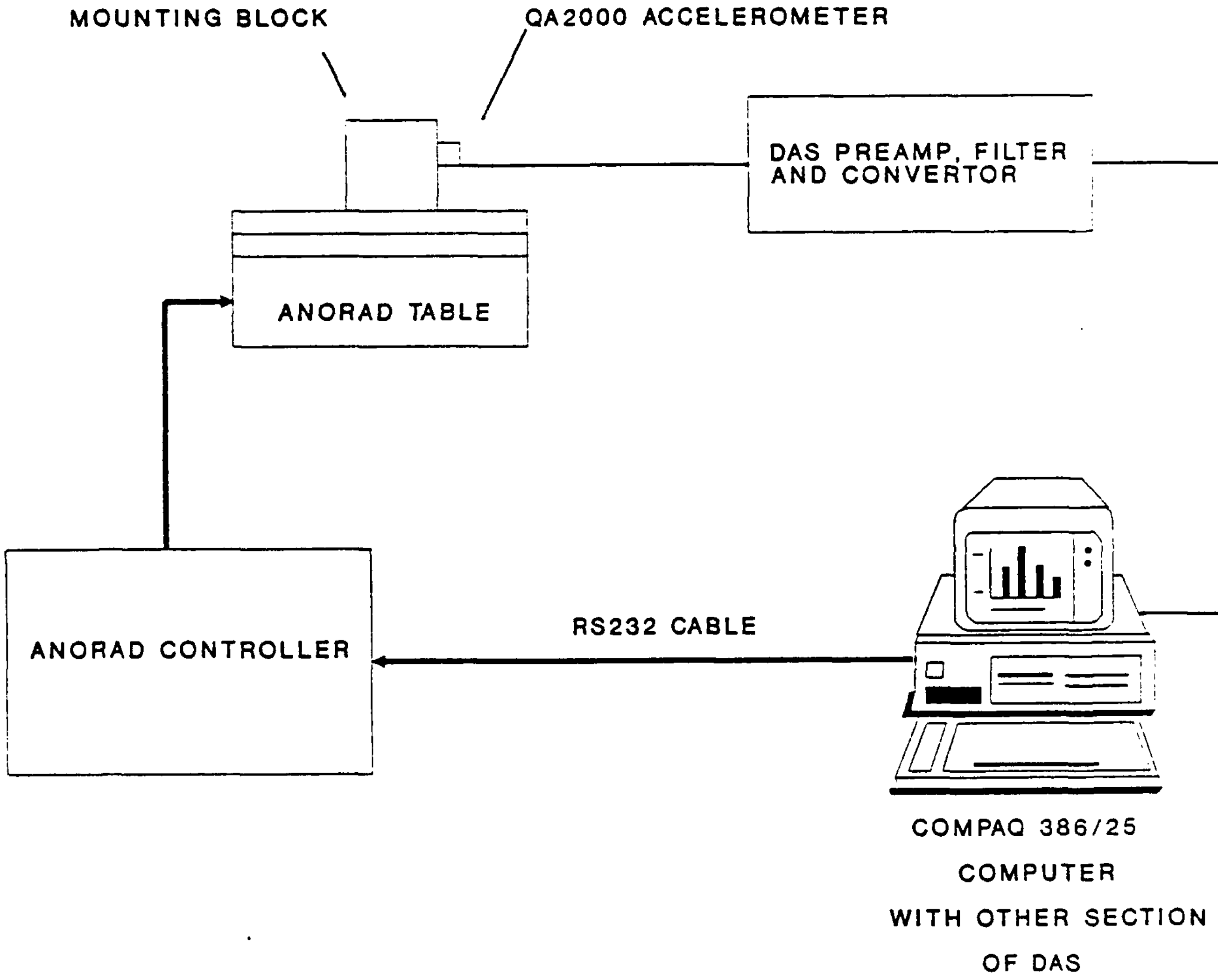

MOUNTING BLOCK
QA2000 ACCELEROMETER
DAS PREAMP, FILTER AND CONVERTOR
ANORAD TABLE
RS232 CABLE
ANORAD CONTROLLER
COMPAQ 386/25 COMPUTER WITH OTHER SECTION OF DAS

Figure 48 - Sketch of System Setup

8.2 Test Results

Using the real-time software the system was tested for each application scenario. The single motion test, multiple motion test with ZUPT, and multiple motion test with ZUPT and two bases (M2) were the only compensation schemes that provided any useful results. Table 10 summurizes the results of these tests. The scheme number in the table is the menu number as specified in section 6. Each result in the table represents the statistics of 50 runs for a filter bandwidth of 40 Hz, an A-to-D range of 1.0 Gs, and the trapezoidal integration method. The log files for the tests are included in Appendix E. The compensation schemes for the remaining tests relied on the accelerometer bias to be the dominant error source and also that the bias could be modeleled by

$$B = B_o + B_1 t \quad .$$

Evidently the accelerometer errors such as SF stability and nonlinearity are as dominant as the bias.

Since the results of these tests were below that expected of the DAS, a more detailed simulation of the system including the accelerometer error sources should be conducted. Such a simulation would be able to determine system performance for a variety of accelerometers, DAS's, and integration algorithms.

Table 10 - Real-Time Test Results

Scheme	Sampling Rate	Max Err	Mean Err	Stand Dev
1	200 Hz	0.268	0.150	0.055
2	200 Hz	-0.412	-0.281	0.049
6	200 Hz	-9.255	-0.383	1.617
6	200 Hz	1.328	0.008	0.219
1	300 Hz	0.187	0.061	0.054
6	300 Hz	-0.583	-0.012	0.092

Position errors and standard deviation are in inches.

9 Conclusions and Recommendations

The test results of the system were below that expected of the DAS. Therefore, the dominant error sources seem to reside in the accelerometer. An accelerometer that is specifically designed for this application (operation range of ± 2 Gs) is needed. Also, from the simulation of the DAS, it was determined that 20 bits resolution is more than needed and a 100 Hz sampling rate is too slow for real-time accuracy. Hence, using a 16-bit converter to sample at 20 kHz and then averaging 4 samples resulting in a 5 kHz acceleration update should be more accurate than the current system for real-time position accuracies less than 10 thousands of an inch. The averaging scheme will also tend to increase the effective resolution of the system. Before continuing any more hardware development an in-depth simulation of the entire system (open and closed loop) from the accelerometer to the processor should be conducted. This approach would better define the propagation of error sources to the position error.

Acknowledgements

We thank Troy Hester of Inertial Systems at MICOM for his input into many technical discussions of the problem at hand. Mark Bailey (SDI) was responsible for developing the software to control the Anorad table and Glenn Cottles (SDI) fabricated the DAS as well as other electronic assemblies.

References

[1] W. E., Industrial Robots: Computer Interface and Control, Prentice-Hall, 1985.

[2] Paul, R.P., Robot Manipulator, MIT Press, 1981.

[3] Hung, J. C., "Inertial Measurement Based Robot Manipulator Control", Proceedings of the First International Symposium on Inertial Technology, Chinese Society of Inertial Technology, Beijing, China, 1989.

[4] Hung, J. C., "Analysis and Software for the Development of An Inertial Aided Robot Manipulator," one of the final technical reports done under U. S. MICOM Contract DAAH01-87-D-0182, 31 January 1989.

[5] "Instruction Manual for Sunstrand Data Control's Q-Flex Accelerometers," Sunstrand Data Control, Inc., Redmond, WA, 1986.

[6] C. Hagan, "Gimbal Calibration Analysis", Final Report H-88-12, Sept. 30, 1988.

Appendix A - AD1170 Data Sheet

ANALOG DEVICES

High Resolution, Programmable Integrating A/D Converter

AD1170

FEATURES

Low Nonlinearity:
- **Integral: ±0.001%**
- **Differential: ±0.00035%**

Microcomputer-Based Design
- **Programmable Integration Time: 1 to 350ms with Resolution from 7 to 18 Bits**
- **Programmable Output Data Format**
- **Auto-Zeroed Operation and Electronic Calibration (No External Trim Potentiometers)**
- **Microprocessor Compatible Interface**

High Throughput: Over 50 Conversions/Second for Line Cycle Integration Period

High Normal Mode Rejection: 54dB at 60Hz

Small Size: 1.24" × 2.5" × 0.55" max

APPLICATIONS

Data Acquisition Systems
Scientific Instruments
Medical Instruments
Weighing Systems
Automatic Test Equipment

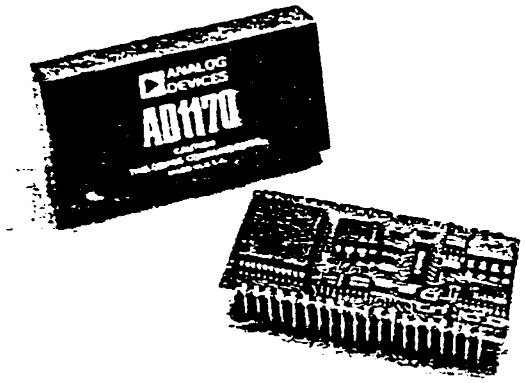
GENERAL DESCRIPTION

The AD1170 is a high resolution integrating A/D converter intended for applications requiring high accuracy and high throughput at low cost. A novel conversion architecture provides the user with outstanding accuracy, stability and ease of use.

The AD1170 is a complete microcomputer-based measurement subsystem, composed of three major elements: a highly precise charge balancing converter, a single chip microcomputer, and a custom CMOS controller chip. The AD1170 offers independently programmable integration time (from one millisecond to 350 milliseconds) and data format (offset binary or two's complement, from 7 to 22 bits). The converter is fully auto-zeroed and exhibits a span drift of only 9ppm/°C, assuring stable, accurate readings.

The AD1170 may be interfaced to any microcomputer based system in a memory mapped or I/O mapped fashion via an 8-bit data bus. The AD1170's advanced features are controlled by simple commands sent to it via this bus.

The converter utilizes surface mount technology and is housed in a small 1.24" × 2.5" × 0.55" package. It operates from ±15V dc and +5V dc power.



PRODUCT HIGHLIGHTS

1. The AD1170, unlike dual slope converters, offers the user the capability of programming the integration time by selecting one of seven preset integration periods or by loading an arbitrary integration period over the interface bus.
2. The AD1170 architecture provides for user programmable data format independent of the integration time. All data is computed to 22-bit resolution and the user may specify any resolution from 7 to 22 bits. Usable resolution will typically be limited to 18-bits due to measurement and calibration noise error.
3. Electronic digital calibration eliminates the need for trim potentiometers. Calibration can be performed at any time by applying an external reference voltage to the input and invoking a calibration command. The calibration data is stored in an internal nonvolatile memory chip.
4. Internal calibration cycles may be programmed to occur whenever the converter is idle, assuring negligible offset drift and only 9ppm/°C span drift.
5. The conversion rate is greater than 50 conversions per second when programmed for 60Hz line cycle integration. The maximum conversion rate is greater than 250 conversions per second, using a one millisecond integration period.

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices.

One Technology Way, P. O. Box 9106, Norwood, MA 02062-9106
Tel: 617/329-4700 TWX: 710/394-6577
West Coast 714/641-9391
Midwest 312/980-0300
Texas 214/231-5094

SPECIFICATIONS (typical @ +25°C, V_S = ±15V, V_B = +5V unless otherwise specified)

MODEL	Min	Typ	Max	Units
RESOLUTION[1]	7		18	Bits
ACCURACY				
Integral Nonlinearity[2]		±0.001		% SPAN
THROUGHPUT RATE[3]				
Time (Integrate) = 1ms	250			conv/S
Time (Integrate) = 16.667ms	50			conv/S
Time (Integrate) = 100ms	9			conv/S
DIFFERENTIAL NONLINEARITY				
T (int) @ T (cal)				
1ms 10ms		±0.01		% SPAN
16.667ms 100ms		±0.0008		% SPAN
300ms 300ms		±0.00035		% SPAN
STABILITY				
Span		±9		ppm SPAN/°C
POWER SUPPLY REJECTION RATIO				
(Span Error vs. Analog Supply Voltage)		60		ppm of Reading/V
INPUT CHARACTERISTICS				
Analog Input Range				
dc	−5		+5	V
dc PlusNormal-Mode Voltge	−6		+6	V
Absolute Maximum (Without Damage)	−30		+30	V
Normal-Mode Rejection				
@ 60Hz		54		dB
@ 50Hz		60		dB
Input Bias Current		10		nA
Input Impedance		100		MΩ
REFERENCE				
Output Voltage		5		V dc
Output Current		2		mA
Input Range	4.5		5.5	V dc
DIGITAL LEVELS				
Inputs				
Low			0.8	V
High	2.0			V
Outputs				
Low (@ 4mA)			0.45	V
High (@ 100μA)	2.4			V
WARMUP TIME				
to 60ppm SPAN		5		min
to 20ppm SPAN		15		min
POWER REQUIREMENTS				
$+V_S$ and $-V_S$	9	15	18	V
$+V_D$	4.75	5	5.25	V
Supply Current Drain				
@ ±15V		12		mA
@ +5V		110		mA
TEMPERATURE RANGE				
Rated Performance	0		+70	°C
Storage	−25		+85	°C
SIZE	1.24" × 2.5" × 0.55" max (31.4 × 63.5 × 14.0mm)			
PRICE	$150 (1-9)		$98 (100s)	

NOTES

[1] The usable resolution is limited by noise, which is largely determined by the integration period and calibration period. Consult the chart in Figure 4 for typical peak-to-peak noise versus integration and calibration period.

[2] The integral linearity is defined as the deviation from a straight line drawn between the endpoints of the converter. This specification is independent of gain and/or offset errors.

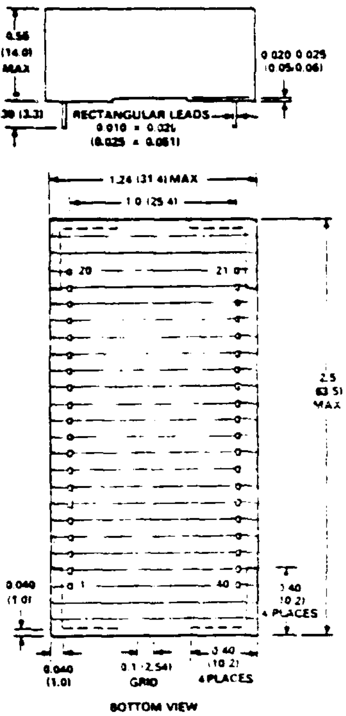
[3] Throughput Rate is calculated by the formula: $\frac{1000}{T(int) + 3\ \text{milliseconds}}$ = minimum conversions/second

Where T (int) is expressed in number of milliseconds

Specifications subject to change without notice

OUTLINE DIMENSIONS

Dimensions shown in inches and (mm).



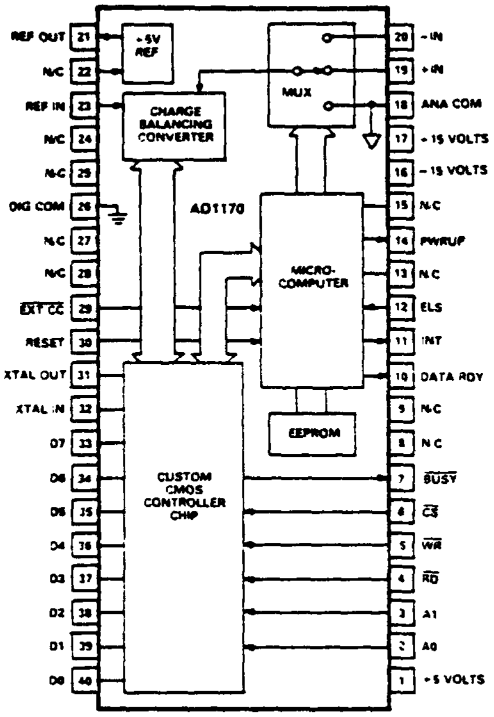
WARNING!

ESD SENSITIVE DEVICE

CAUTION: OBSERVE PROPER PLUG-IN POLARITY TO PREVENT DAMAGE TO CONVERTER

PIN DESCRIPTIONS

PIN	SIGNAL	DESCRIPTION
1 ✓	+5V	Digital Power Supply
2,3 ✓	A0 A1	Address Control Lines
4 ✓	$\overline{RD}$	Read Data Strobe
5 ✓	$\overline{WR}$	Write Data Strobe
6 ✓	$\overline{CS}$	Chip Select
7 ✓	$\overline{BUSY}$	When Low, Indicates Device Busy When High, Indicates Device Ready for Command
10 ✓	DTA RDY	When High, Indicates That Data From Most Recent Conversion Command is Ready
11	INT	When High, Indicates Device is Currently Integrating Input Signal. Goes Low to Indicate Integration Complete
12	ELS	External Line Sample Input (Used with ELS Command to Sense an Externally Provided Sample of the Line Frequency
14	PWR UP	When High, Indicates Power Up Initialization in Progress
16	−15V	Negative Analog Power Supply
17	+15V	Positive Analog Power Supply
18	ANA COM	Analog Common (the Reference Point for Analog Power Supplies
19	+IN	Positive Signal Input
20	−IN	Negative Signal Input
21	REF OUT	Internal +5V Reference Output
23	REF IN	Reference Input Normally Connected to Ref Out
26 ✓	DIG COM	Digital Common (the Reference Point for the Digital Power Supply
29 ✓	$\overline{E_{xt}CC}$	External Convert Command Input
30	RESET	Reset Input. Usually Connected to an RC Network for Automatic Reset Upon Power Up
31,32	XTAL OUT, XTAL IN	Connections for 12MHz Crystal (Series Resonant 30Ω ESR). Alternatively, Xtal In May Be Driven From an External 12MHz Logic Signal
33-40	D7 D0	Bidirectional Data Bus
8, 9, 13, 15, 22, 24 25, 27 28		DO NOT CONNECT



LEAVE N/C PINS UNCONNECTED

Functional Block Diagram

FACTORY DEFAULT SETTINGS

The AD1170's internal nonvolatile memory stores various A/D parameters as programmed by the user (such as the integration period, output data format, calibration coefficient, etc.). The AD1170 is calibrated at the factory with the following default settings:

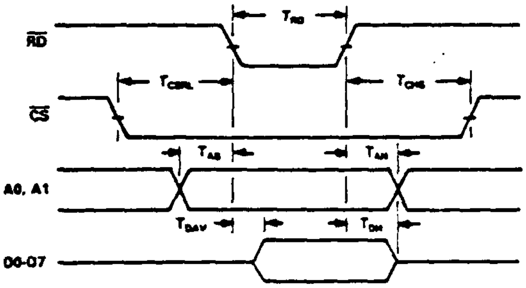
FORMAT: 16-bit, offset binary
DEFAULT T(int): 16.667 milliseconds
(code 2)
DEFAULT T(cal): 100 milliseconds
(code 4)

AD1170 ARCHITECTURAL OVERVIEW

The AD1170 is a complete microcomputer-based measurement subsystem, containing three major elements: a highly precise charge balancing converter, a single chip microcomputer, and a custom CMOS controller chip.

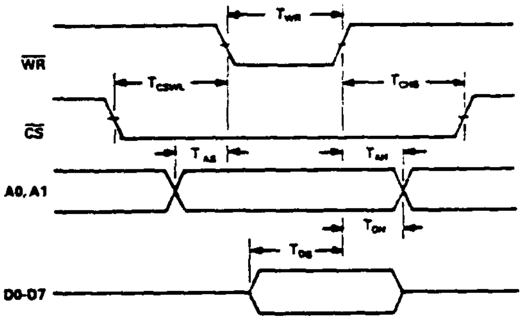
The heart of the measurement technique is the charge balancing converter (essentially a voltage to frequency converter). This converter measures the input signal by balancing a proportional current against a train of precisely controlled reference current pulses using an integrator. The microprocessor, together with the counting and gating circuitry within the CMOS controller chip, measures the period of the reference current pulses by interpolating them using a 12MHz clock signal. The resulting data is converted to binary representation by the use of floating point firmware routines within the microprocessor.

When the AD1170 is triggered to perform a conversion, two separate phases are performed: first, an integration phase, where the input signal is actually measured, and then a computation phase, where the data from the integration phase is processed, along with both the volatile and nonvolatile calibration data, and formatted for output as the user desires.



READ CYCLE TIMING REQUIREMENTS

PARAMETER	DESCRIPTION	MIN	TYP	MAX	UNITS
T_{RD}	RD Pulse Width	150			ns
T_{CSRL}	Chip Select to RD Low	0			ns
T_{CHS}	Chip Select Hold Time	0			ns
T_{AS}	Address Setup Time	10			ns
T_{AH}	Address Hold Time	0			ns
T_{DAV}	Data Valid Time			100	ns
T_{DH}	Data Hold Time			80	ns



WRITE CYCLE TIMING REQUIREMENTS

PARAMETER	DESCRIPTION	MIN	TYP	MAX	UNITS
T_{WR}	$\overline{WR}$ Pulse Width	100			ns
T_{CSWL}	Chip Select to $\overline{WR}$ Low	0			ns
T_{CHS}	Chip Select Hold Time	0			ns
T_{AS}	Address Setup time	10			ns
T_{AH}	Address Hold Time	0			ns
T_{DS}	Data Setup Time	80			ns
T_{DH}	Data Hold Time	20			ns

Figure 1. Timing Diagrams and Requirements

The duration of the integration phase can be programmed by the user, and may be as short as one millisecond, or as long as 350 milliseconds. The computation phase always lasts approximately three milliseconds and commences immediately after the integration phase is over. Therefore, the total conversion time will equal the user programmed integrate time plus a fixed 3 milliseconds. Status signals are provided to indicate when the data is ready and when the converter may be retriggered for the next conversion.

For maximum stability, the AD1170 periodically calibrates itself by performing measurements upon a zero input signal and a full-scale signal provided by the internal reference. This technique cancels any drift within the charge balancing converter itself, resulting in negligible offset drift, and gain stability equal to that of the reference. Calibration cycles may be programmed to take place whenever the AD1170 is idle, or they may be invoked under system control.

he AD1170 contains no internal trims; its span accuracy is ctory calibrated by using the ECAL (Electronic CALibration) ature. This feature is a firmware routine which measures an ternally applied reference voltage, compares it to the internal ference voltage, and computes a span correction factor which stored in nonvolatile memory. The correction factor is then pplied to all subsequent measurements, thereby compensating r the reference error. The ECAL function may be invoked by e user at any time, thereby replacing the usual trim potentiometer ith a totally electronic calibration capability.

NDERSTANDING THE AD1170 SPECIFICATIONS

he AD1170, because of its unique conversion technique, is pecified quite differently from more conventional integrating onverters. The following sections will help the user to understand here the sources of error are, and how to extract the best ossible performance from the converter.

here are two primary sources of error in the AD1170: integral onlinearity of the charge balancing converter, which influences ll conversions equally, regardless of the integration period and alibration period; and the noise error of the measurement/calibration process, which is a function of the integration period nd calibration period as selected by the user.

NTEGRAL NONLINEARITY

The integral nonlinearity of the charge balancing converter CBC) is ±10ppm (±0.001%) of Span. This specification is an "endpoint" nonlinearity measurement; i.e., the typical deviation seen from a straight line drawn between the CBC output at −5 volts and its output at +5 volts. This specification excludes any gain or offset error.

If the converter was externally calibrated at its end points (−5 volts and +5 volts), then the integral nonlinearity would also be the relative accuracy of the converter. This is not the case in the AD1170, however, because calibration is performed internally at 0 and +5 volts, rather than −5 and +5 volts. This calibration technique, superimposed upon the integral nonlinearity of the CBC, results in the curve shown in Figure 2.

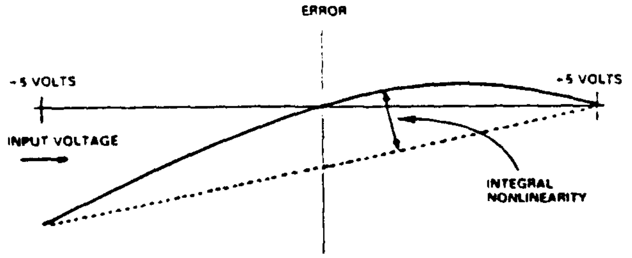


Figure 2. Relative Accuracy and Integral Nonlinearity when Calibrated

As shown in the diagram, the calibration technique tends to exaggerate the relative error at the negative end of the scale, and reduce the error between 0 and +5 volts. This characteristic happens to be most beneficial when using the AD1170 in systems where the input comes from a sensor whose signal is mostly positive, such as a thermocouple.

For systems where the user desires to minimize the relative error equally across the whole span of the converter, it is possible to intentionally introduce a span error during the ECAL procedure, as shown in Figure 3. This scheme sacrifices positive full-scale accuracy in order to minimize negative full scale error. The net result is a relative accuacy equal to the integral nonlinearity.

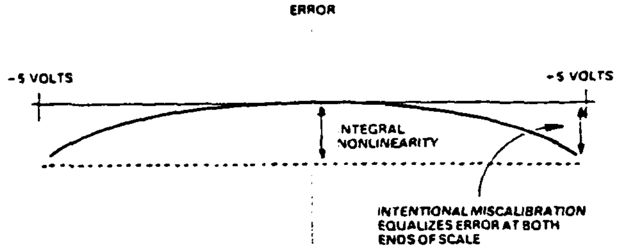

ERROR
-5 VOLTS
+5 VOLTS
INTEGRAL NONLINEARITY
INTENTIONAL MISCALIBRATION EQUALIZES ERROR AT BOTH ENDS OF SCALE

Figure 3. Relative Accuracy with Intentional Span Error at +F.S.

In both cases the accuracy of the input offset (which is servo controlled) is not compromised.

MEASUREMENT/CALIBRATION NOISE

Measurement noise refers to the conversion-to-conversion uncertainty caused either by mathematical truncation or device noise.

Calibration noise is actually the measurement noise resulting from the calibration process. The converter stabilizes itself by performing internal measurements of the reference, and of ground; these measurements have the same uncertainty due to noise as does the normal measurement process.

The measurement and calibration noise error of the AD1170 determines the differential linearity, or useable resolution, of the converter. This parameter determines the usable resolution because it defines what codes can be seen through the noise. The specified value is the amount of error, in either direction from the average reading, which will not be exceeded for 95% of all conversions. This parameter, as in all integrating converters, is a function of the integration time; long conversions result in very high resolution, while short conversions provide lower resolution. In the AD1170, all internal computations are always carried out to 22-bit resolution, but useable resolution is limited by the peak-to-peak noise, as determined by T(cal) and T(int).

The chart shown in Figure 4, illustrates the typical peak-to-peak noise (in ppm Span) versus T(int) and T(cal). These numbers can be used to indicate how much useable resolution can be

	T(cal) =	1ms	10ms	16.7ms	20ms	100ms	166.7ms	300ms	CAL DISABLED	UNITS
T(int) =	1ms	208	115	115	114	113	112	111	110	±ppm of SPAN
	10ms		24	18	16	13	13	13	12	
	16.7ms			14	13	8	8	8	8	
	20ms				12	7	7	7	7	
	100ms					4.0	4.0	3.5	3.5	
	166.7ms						4.0	3.5	3.5	
	300ms							3.5	3.5	

Figure 4. Typical Peak-to-Peak Noise (in ppm Span) Versus T(int) and T(cal)

expected under a given set of operating conditions. For example, a peak-to-peak noise of ±8ppm is approximately analogous to a flicker of ±0.5LSB at 16 bits of resolution. Under these conditions, a user could set the default format for the AD1170 to 16-bit resolution, and observe no more than ±1/2LSB of differential error. See Table I for conversion of typical peak-to-peak noise to Differential Nonlinearity and Useable Resolution.

The chart in Figure 4 may also be used to determine the minimum effective calibration time for a specified integration period; the noise contributions of both the measurement cycle and the calibration cycle combine as the "root sum square", and the combined effect tends to asymptotically approach a baseline value determined by the shorter of the two. For example, a T(cal) greater than 10 milliseconds does little or nothing to improve the noise performance for conversions using a T(int) of 1 millisecond.

NOISE (ppm Span)	RESOLUTION AT 1/2LSB DNL ERROR (NO. OF BITS)	RESOLUTION AT 1LSB DNL ERROR (NO. OF BITS)	DIFFERENTIAL NONLINEARITY (% Span)
244	11	12	0.024
122	12	13	0.012
61	13	14	0.006
31	14	15	0.003
15	15	16	0.0015
8	16	17	0.00076
4	17	18	0.00038
2	18	19	0.00019

Table I. Conversion of Noise Error to DNL and Usable Resolution

SIGNAL INPUT CONNECTIONS

The AD1170 has both a positive input pin (+IN) as well as a negative input pin (−IN), but the AD1170 input is not a true differential input. The negative input pin is an input used during calibration cycles to establish the zero reference. In applications with static ground offsets, the −IN pin may be used as a ground sense input, to sense a signal reference point which is offset from analog common by a small differential. Both the −IN and +IN signals must have a bias current path back to analog common. Figure 5 illustrates the proper use of the input signal connections.

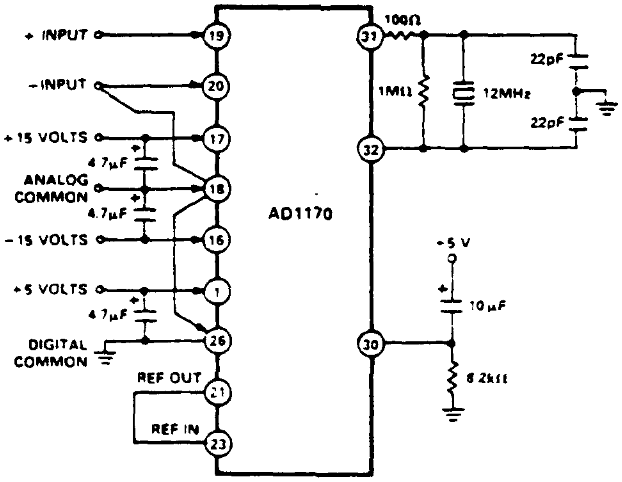


Figure 5. Input, Power, Reset, and Clock Connections

RESET

A reset sequence must be accomplished after power-up and before any access to the converter. The RESET line initializes the internal logic of the AD1170. This line may be driven from an external source, such as may exist in most computer based systems, or it may be connected to a simple RC circuit which will automatically invoke a reset sequence at power-up. Figure 5 illustrates the recommended circuit.

When driving the RESET line from an external source, the line must be held high for at least 2 microseconds after the oscillator is running and stable (typically 300 microseconds after power is applied) in order to assure a proper reset.

CLOCK

The AD1170 requires a 12MHz clock for operation. This clock may be supplied by connecting the XTAL OUT and XTAL IN pins to a 12MHz crystal, along with two resistors and two capacitors as shown in Figure 5.

The user may also supply a 12MHz logic signal from an external source, such as may be available in the user's system. In this case, the external clock should be applied to the XTAL IN pin, and the XTAL OUT pin should remain unconnected.

POWERING THE AD1170

For best performance, the user should pay careful attention to proper power supply bypassing, as well as grounding. Analog common and digital common are not connected internal to the module, but must be connected externally. Figure 5 illustrates the proper connection of power and ground to the AD1170[1].

REFERENCE CONNECTIONS

The internal +5 volt reference of the AD1170 is brought out to Pin 21 of the module; for normal operation, it should be connected to the reference input (Pin 23).

An external reference voltage of from 4.5 to 5.5 volts may be applied to the reference input (Pin 23), and the reference output may remain unconnected. The data will be ratiometric to that reference. The input impedance of the reference input is approximately 16K ohms. The reference input is not dynamic; any external reference voltage must be an essentially static DC signal.

INTERFACING TO THE AD1170

The AD1170 contains an eight-bit microprocessor compatible interface structure, including control lines. It can be interfaced to any microprocessor-based system in either a memory mapped or I/O mapped mode, and occupies four successive bytes of read/write address space, as shown in Figure 6.[1]

	$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A1	A0	FUNCTION
	H	X	X	X	X	Device Not Selected
WRITE	L	H	L	H	H	(Unused)
	L	H	L	H	L	Parameter 2 Write
	L	H	L	L	H	Parameter 1 Write
	L	H	L	L	L	Command Write
READ	L	L	H	H	H	High Data Read
	L	L	H	H	L	Mid Data Read
	L	L	H	L	H	Low Data Read
	L	L	H	L	L	Status Read

X = DON'T CARE X = DON'T CARE

Figure 6. Control Functions

[1] Attempting to READ and WRITE at the same time $\overline{RD}$ and $\overline{WR}$ set low may alter the contents of the internal nonvolatile memory.

he AD1170 is controlled by writing a command into the lowest yte of the device image. Upon receipt of the command byte, ie BUSY line is set low, indicating that command interpretation is in progress. The BUSY line returns high, following command nterpretation and a command dependent execution time. This ignals that the command execution has been completed, and nother command may now be written. The logical inverse of he BUSY line is available in the STATUS byte for use in polling. ee the section below about THE STATUS BYTE.

Vhen the command requires one or two parameters, in addition o the command byte, they must be written into the second and hird parameter bytes of the image *before* writing the command yte. This is because writing the command byte triggers the nicroprocessor to begin command interpretation.

ollowing the execution phase of any command, the CMD ERR it in the STATUS byte will indicate acceptance or rejection of he command. When set, this bit indicates that there was a ontextual or syntactic error in the command or parameters.

Conversions may be initiated either by bus command, or by a igh to low transition of the EXT CC line[1]. Externally triggered onversions behave in the same way as bus triggered conversions, xcept that the BUSY line and the BUSY bit in the status word emain inactive; the end of execution of externally triggered onversions must be determined by examination of the DTA RDY line or the DTA RDY bit in the STATUS word.

THE STATUS BYTE

The lowest readable byte of the device image is the STATUS byte; it contains six bits of information about the current status of the AD1170. This byte may be examined by the host processor at any time. The individual bits in the status byte (see Figure 7) are assigned the following functions:

BIT0 The BUSY bit is an inverted version of the signal on Pin 7 of the module. When low, it indicates that the AD1170 is ready to receive a command. When high, it indicates that the AD1170 is busy executing the last command. Any commands loaded while the BUSY signal is high will be ignored.

BIT1 The DTA RDY bit (also available on Pin 10 of the module) goes high to indicate that the data from the most recent conversion is available in the LOW DATA, MID DATA, and HIGH DATA registers. This bit is cleared at the start of the next conversion. It may also be cleared by executing an EOI command.

BIT2 The DATA SAT bit is set by any conversion which is saturated, i.e.. any conversion whose output data exceeds positive or negative full scale.

BIT3 The CMD ERR bit indicates that the most recently loaded command contained a contextual or syntactic error, or was not recognized. It is cleared when the next command is loaded.

BIT4 The INT bit (also available on Pin 11 of the module) goes high to indicate that the input signal is currently being integrated. It is used in multiplexed systems to determine when the input multiplexer may be switched.

BIT5 The PWRUP bit (also available on Pin 14 of the module) goes high when the module is powered up or when the RST command is executed. It remains high until device initialization is complete. This signal is used to indicate readiness of the converter after system initialization.

B7	B6	B5	B4	B3	B2	B1	B0
*	*	PWRUP	INT	CMD ERROR	DATA SAT	DATA RDY	BUSY

* UNUSED: CONTENTS INDETERMINATE

Figure 7. The Status Byte

OUTPUT DATA FORMAT

The AD1170 architecture allows a programmable data format independent of the integration time. The user may specify any resolution from 7 to 22 bits, and may specify either offset binary coding or two's complement coding. Programming the data format is accomplished via the use of the SDF command, using the format code described in the table in Figure 8 as the PARAMETER 1 value.

C_4	C_3	C_2	C_1	C_0	DATA FORMAT
H	X	X	X	X	Two's Complement
L	X	X	X	X	Offset Binary
X	H	H	H	H	22 Bits
X	H	H	H	L	21 Bits
X	H	H	L	H	20 Bits
X	H	H	L	L	19 Bits
X	H	L	H	H	18 Bits
X	H	L	H	L	17 Bits
X	H	L	L	H	16 Bits
X	H	L	L	L	15 Bits
X	L	H	H	H	14 Bits
X	L	H	H	L	13 Bits
X	L	H	L	H	12 Bits
X	L	H	L	L	11 Bits
X	L	L	H	H	10 Bits
X	L	L	H	L	9 Bits
X	L	L	L	H	8 Bits
X	L	L	L	L	7 Bits

X = DON'T CARE ($C_7 C_6 C_5$ = X FOR ALL DATA FORMATS)

Figure 8. Format Code

It should be noted that the AD1170 computes all data to 22 bit resolution. However, not all 22 bits are useable, since the differential performance is largely dependent upon factors such as integration period and calibration period. The SDF command simply serves to round off the result to the specified number of bits. The graph in Figure 4 can be used to estimate the amount of useable resolution achievable for a specified integration period and calibration period.

The output data is always right justified within the three output bytes (LOW DATA, MID DATA, and HIGH DATA). If two's complement format is selected, the MSB of the data is inverted and extended all the way to the top of the HIGH DATA byte. For example, if 16 bit two's complement format is selected, the data will appear in the LOW DATA and MID DATA bytes, and the MSB will be 0 for positive inputs.[2] The format is a nonvolatile parameter; whenever an SAVA command is executed, the current format will be saved to nonvolatile memory, and will become the default format upon powerup.

[1] The minimum duaration for EXT CC is one microsecond.

[2] Since the sign is extended all the way to the top of the uppermost byte, the HIGH DATA byte will be filled with the value of the MSB.

PROGRAMMING THE INTEGRATION PERIOD

The key parameter of any integrating A/D converter is the integration period. As shown in Figure 9, an integrating A/D converter provides maximum normal mode rejection at those frequencies which are integral multiples of 1/T(int), where T(int) is the integration period. The most common way to exploit this characteristic is to set the integration period equal to one period of the power line frequency so that ac hum will be rejected.

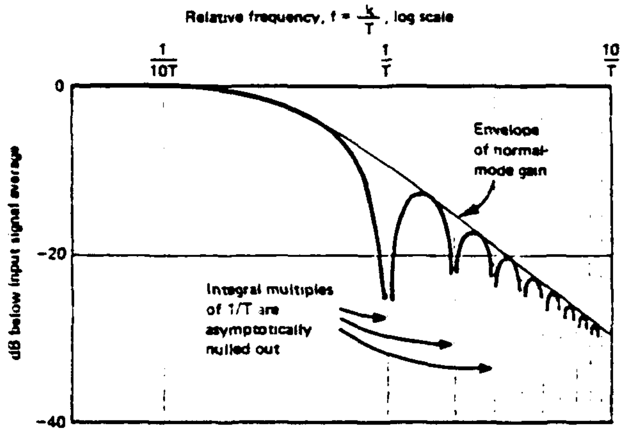


Figure 9. Normal Mode Rejection

The duration of the integration also affects the resulting resolution of the data; long integration times result in more usable resolution than do short integration periods.

The AD1170, unlike most dual slope converters, offers the user the capability of programming the integration time. This feature can be used to great advantage in systems design, since the integration time can be optimized for differing system conditions. For example, in systems whose inputs are severely polluted by 60Hz noise, the user may wish to program the AD1170 for a 100 millisecond integration time, which will result in excellent 60Hz normal mode rejection. In another application, a user may wish to scan a large number of channels rapidly, looking for gross input changes, then slow down in order to make a high resolution conversion before resuming rapid scanning.

The AD1170 offers the user a number of different ways to set the integration period. The simplest way is by using the SDI command to set the default integration period to one of seven preset periods (1ms, 10ms, 16.66ms, 20ms, 100ms, 166.66ms, 300ms). The first two preset periods offer fairly rapid scanning at reduced resolution; the other five represent American and European line voltage standards or multiples thereof. For single conversions without altering the default integration time, the CNVP command may be used, which also allows the selection of one of these seven preset periods. These preset periods and their corresponding codes are listed in the table of Figure 10.

Another way in which the integration period may be programmed is via the EIS command, which allows the user to load the externally definable period register with a binary value[1] proportional to the desired integration period. Using this technique, the user may specify any period from one millisecond to 350 milliseconds (with 200 microsecond accuracy). Access to this user definable period is via the SDI or CNVP commands; the last entry in Figure 10 is used to select the period defined by the EIS or ELS command.

C_2	C_1	C_0	INTEGRATION TIME	NOTES
L	L	L	1 Millisecond	
L	L	H	10 Milliseconds	
L	H	L	16.667 Milliseconds	1 cycle @ 60Hz
L	H	H	20 Milliseconds	1 cycle @ 50Hz
H	L	L	100 Milliseconds	50/60Hz
H	L	H	166.67 Milliseconds	10 cycles @ 60Hz
H	H	L	300 Milliseconds	50/60Hz
H	H	H	(See Note)	

NOTE
This code is used for externally loaded integration times (defined with the EIS Command) or externally measured times (from the ELS Command). The value can be anywhere from 1 Millisecond to 350 Milliseconds.

Figure 10. Preset Integration Periods

The third way to set the integration period is via the external line sampling feature, using the ELS command. This command samples the period of the logic signal presented to the ELS input pin (Pin 12), and sets the externally definable period register accordingly. This feature is most useful in environments with fluctuating line frequencies. By executing an occasional ELS command, the converter effectively "tracks" the line frequency. To use this feature, a clean, bounce free logic representation of the line frequency must be present at the ELS input during the execution of the ELS command. Once having performed the ELS command, the measured integration time may be selected using the SDI or CNVP commands along with the (HHH) code from the table in Figure 10[2].

It should be noted that the actual integration period used in the measurement process is accurate to about ±200μs, due to the limitations of the charge balancing converter. This is adequate, however, for greater than 50dB of normal mode rejection at 60Hz when using an integration period of 1/60 second. Even greater normal mode rejection may be obtained when the integration period is a multiple of the line frequency period.

CONTROLLING THE CALIBRATION CYCLE

The AD1170 achieves its excellent span and offset stability by calibrating itself against its internal reference voltages. The user can control the frequency of occurrence for calibration cycles and their duration.

The duration of the calibration cycle is an important parameter, because it affects the accuracy of the calibration cycle itself. Errors in the calibration cycle appear in the output data as instantaneous offset and span errors. If automatic "background" calibration is enabled, these errors effectively appear as noise. Just as in the case of input conversions, longer calibration times result in more accuracy and less noise.

Of course there may be system applications where there simply isn't sufficient time to perform a long calibration cycle. For this reason, the AD1170 offers the user the ability to specify the calibration period, using the SDC command.

[1] See the section titled "The AD1170 Command Set" for the formula used to compute the proper binary value.

[2] Caution is advised: if no signal is present at the ELS input when the ELS command is executed, or if the signal is not within acceptable frequency limits, the module may "hang" and require a hardware reset to continue operation.

ie argument for the SDC command is the same three-bit code is used for the SDI and CNVP commands. The reason for is is that each calibration cycle consists essentially of two dinary conversion cycles, performed upon the internal zero d span references. For example, if an SDC command with an gument of 3 is executed, the default calibration time will en be approximately 49 milliseconds (two conversions of 20 illiseconds plus approximately 9 milliseconds for the internal athematics).

he user may also disable or enable background calibration. In stems where the AD1170 may be periodically idle, i.e., not erforming input conversions, background calibration is a good ioice. This mode is enabled with the CALEN command and ill cause the AD1170 to continually initiate an internal calibration ycle whenever the converter is otherwise unoccupied. Any onversion commands received during a cal cycle will cause that il cycle to be aborted in favor of the input conversion, thereby iving the user priority over calibration. This mode of operation automatic and transparent.

he CALDI instruction is used to disable background calibration. Vhen this instruction is executed, the converter will be completely dle between convert commands, and calibration cycles will only ccur when invoked by the SCAL command. This mode of peration is best when the user would like to perform input onversions at the maximum rate, and/or when the system affords specific convenient time to perform calibration.

There are no hard and fast rules about the best way to apply all of this flexibility, but best performance will be obtained if the following points are observed:

- Consult the chart in Figure 4 to determine the minimum effective calibration period for use with a desired integration period.
- Don't use automatic background calibration unless your system will allow the converter enough uninterrupted time to perform at least one calibration cycle. For example, if you are using a calibration period code of 3, your system must periodically allow at least 49 milliseconds without a convert command or calibration will not occur.
- Remember that the purpose of the calibration cycle is to cancel the intrinsic drift of the charge balancing converter within the AD1170 itself. If the converter is in a stable environment, calibration may be done less frequently. The best possible performance will be achieved in stable ambient temperatures, where calibration is manually invoked by the system at relatively long intervals, using the longest allowable calibration time.
- Very short calibration times, although allowed by the AD1170 firmware, are not especially useful because they introduce more error than they compensate. The only useful purpose of very short calibration times is in systems which are operating in rapidly changing ambient temperatures, and then only for relatively low resolution conversions.

COMPENSATION OF EXTERNAL OFFSETS

An electronic "null" feature compensates for offset errors of signal conditioning stages preceding the AD1170.

The null feature comprises three commands: NULL measures the input signal (using the current integration time) and stores it in internal RAM; NULEN subtracts the measured value from all subsequent conversions; NULDI cancels the NULEN command's effect.

The sum of the offset value plus the full-scale input should be less than the ±6 volts linear input range of the AD1170. The offset value to be nulled should ideally be no more than a few hundred millivolts in amplitude.

The NULL command does not need to be executed every time the AD1170 is powered up. Since the value measured by the NULL command is saved and restored by the SAVA and RESA commands, the value of the null will be the one saved during the last SAVA command. Execute a NULL command only when a new null measurement is desired.

When NULEN is in effect, the length of each conversion will be extended by approximately 700 microseconds.

ELECTRONIC CALIBRATION

The AD1170 contains an Electronic CALibration capability, which, along with the internal nonvolatile memory chip, effectively eliminates the need for trim potentiometers of any kind. This capability, abbreviated as ECAL, should not be confused with the internal background calibration cycles. ECAL is a completely distinct function used to calibrate the AD1170 to an external reference standard.

The ECAL function measures the ratio of the internal reference voltage in the module with respect to an externally applied reference voltage. The resulting coefficient is applied to the math computations for all subsequent conversions, effectively compensating the module for absolute value errors in its own reference. The ratio is stored in random access memory until the user invokes a SAVA command, which will save this coefficient (along with the other nonvolatile parameters) in the nonvolatile memory chip. When the module is powered up, the previously saved coefficient is recalled from nonvolatile memory and stored in random access memory.

In order to use the ECAL command, the input to the AD1170 must first be presented with an external +5 volt reference standard such as is usually found in many calibration labs. The ECAL command may then be invoked; the external reference voltage must remain at the input until command execution is complete. If the calibration is to be made nonvolatile, a SAVA command must then be invoked.[1]

ECAL may also be used as a means of making limited ratiometric measurements. For example, in some applications, it may be useful to be able to measure the output of some transducer with respect to its excitation; if the excitation can be scaled to the range of 4.5 to 5.5 volts, then it can be used as an excitation for the ECAL process. Having digitized the excitation, all subsequent conversions will be ratioed to the ECAL value. For example, if an ECAL procedure is performed upon a 4.5 volt source, and the converter subsequently digitizes a 2.25 volt signal, the converter output will be half of full scale, or 11000... (assuming offset binary coding). The converter can be restored to absolute calibration by executing a RESA command, which will restore the last nonvolatile ECAL coefficient to random access memory.

The user is cautioned that the nonvolatile memory used in the AD1170 has a finite endurance of 1000 write cycles minimum. Assuming that the AD1170 is calibrated weekly, this implies a device life span of greater than 19 years. Less frequent calibrations mean a proportionately longer life span. This means ECAL may be performed any number of times, but the user should limit the number of SAVA commands in order to extend the life span of the nonvolatile memory.

[1] Since the SAVA command saves all nonvolatile parameters, the user should be sure that the other default parameters, such as integration time and data format, are set to their desired values before SAVA is invoked.

NONVOLATILE MEMORY

The internal nonvolatile memory in the AD1170 is used to store the various nonvolatile parameters associated with A/D operation (for example, the integration period, data format, ECAL coefficient, etc.).

In addition, eight 16-bit words of the nonvolatile memory are made available to the user for general purpose use. They may be accessed using the RDNV and WRNV commands. Because the nonvolatile memory is specified for a finite endurance of 1000 write cycles minimum, it is best used for data which does not regularly need to change, such as configuration information or system calibration parameters.

FACTORY DEFAULT SETTINGS

The AD1170 is calibrated at the factory; the factory default settings are:

Format: 16-bit, offset binary
Default T(int): 16.667 milliseconds (code 2)
Default T(cal): 100 milliseconds (code 4)

THE AD1170 COMMAND SET

The AD1170 command code set includes 20 different functions. Some of the commands require no parameters, while others require one or two parameters which must be loaded into the PARAMETER 1 and PARAMETER 2 registers prior to loading the command register. Some commands (for example, CNVP) have their option parameter embedded in the lowest three bits of the command itself.

The execution time for any command depends on the command. Figure 11 is a synopsis of the available commands, as well as estimates of their execution times.

Each of the commands described below is preceded by an opcode name, along with the digital code (in binary).

CALEN 10110000

CALEN (CALibration ENable) enables automatic background calibration cycling. In this mode, background calibration cycles are executed automatically whenever the AD1170 is not otherwise occupied. If a command is received during a calibration cycle, that cycle will be aborted and the command will be executed.

CALDI 10111000

CALDI (CALibration DIsable) disables automatic background calibration. After executing this command, the AD1170 will be completely idle between commands. While in this state, a single calibration cycle may be invoked with the SCAL command.

CNV 00001000

CNV (CoNVert) causes a single conversion to be performed, using the current default integration time and data format.

CNVP 00010$C_2C_1C_0$

CNVP (CoNVert using specific Preset time) causes a single conversion to be performed, using one of the eight preset integration times as listed in Figure 10. The default integration time is not changed. The three bit code for the desired integration time is embedded in the lowest three bits of the command code.

ECAL 00011000

ECAL (Electronic CALibration) causes an electronic calibration cycle to be performed. An external +5 volt reference voltage must be presented to the input before this command is executed, and the input must remain stable until the end of command execution is signaled by the BUSY line or the BUSY bit in the status word. The calibration data computed by this command is applied to all subsequent conversions, but is not made nonvolatile until a SAVA command is performed.

MNEMONIC	FUNCTIONAL DESCRIPTION	EXECUTION TIME (APPROX)
CNV	Perform a Single Conversion Using the Default Integration Time	T(int) + 3ms
CNVP	Perform a Single Conversion Using the Specified Integration Time	T(int) + 3ms
ELS	Measure Period of Signal at the ELS Input	2 × T(int) + 20ms
ECAL	Perform Electronic CALibration Routine	1.5 seconds
SDI	Set Default Integration Time for Input Measurements	150µs
SDC	Set Default Calibration Period	160µs
SDF	Set Default Data Format	140µs
RESA	Restore All Nonvolatile Parameters from Memory	2.3ms
SAVA	Save All Nonvolatile Parameters to Memory	150ms
WRNV	Write a Word to the User EEPROM Area	22ms
RDNV	Read a Word from the User EEPROM Area	600µs
EOI	Clear the Data Ready Flag	260µs
SCAL	Perform a Single Cal Cycle	2 × T(cal) + 9ms
CALEN	Enable Background Calibration	300µs
CALDI	Disable Background Calibration	310µs
EIS	Set Integration Time to Arbitrary Value	130µs
RST	Reset AD1170 to Power Up Conditions	210ms
NULL	Measure the Offset Voltage Value at the AD1170 Input and Store	T(int) + 3ms
NULEN	Subtract NULL Measured Value from All Subsequent Conversions	250µs
NULDI	Cancel the Effect of the NULEN Command	250µs

EOI 10001000

EOI (End Of Interrupt) clears the DTA RDY bit in the status byte, as well as the DTA RDY line (Pin 10). It is provided as a means of clearing the interrupt source in systems which use an interrupt upon data ready.

ELS 00100000

ELS (External Line Sample) measures the period of the logic signal applied to the ELS input (Pin 12)[1]. This period is loaded into the register associated with the last entry of the table in Figure 10. Input conversions using this measurement as the integration period may be performed by invoking a CNVP command, or by setting the default integration period with the SDI command. This command is intended for use in environments with varying line power frequency; periodically invoking this command allows effective tracking for improved normal mode rejection.

EIS 00101000

EIS (External Integration Set) is used to establish an arbitrary integration period from 1 millisecond to 350 milliseconds. To use this command, first load the PARAMETER 1 and PARAMETER 2 registers with the 16-bit binary number N, which is calculated using the following expression:

$$N = 2^{16} - T(int)/21.333E\text{-}6$$

After the low and high bytes representing N are loaded into the PARAMETER 1 and PARAMETER 2 registers respectively, execute the EIS command. Once this command is executed, the externally loaded integration time can be used via the CNVP or SDI commands.

RESA 01101000

RESA (REStore All) restores all configuration parameters (default integration time, default calibration time, data format, EIS/ELS period, NULL value and electronic calibration data) from nonvolatile memory. After executing this function, all parameters will be restored to their last value as saved by the SAVA command.

SAVA 01001000

SAVA (SAVe All) saves all programmable attributes (default integration time, default calibration time, data format, EIS/ELS period, NULL value and electronic calibration data) into nonvolatile memory.

SDI $00111C_2C_1C_0$

SDI (Set Default Integration time) sets the default integration time to one of the eight preset times listed in Figure 10. The three-bit code for the desired integration time is embedded in the lowest three bits of the command code.

SDF 00110000

SDF (Set Default Format) sets the default data format according to the five bit code loaded into the PARAMETER 1 register prior to execution of this command. The table in Figure 8 illustrates the construction of the five bit code according to the desired data format and resolution.

SCAL 11000000

SCAL (Single CALibration) performs a single background calibration cycle. This command is intended for use when automatic background calibration has been disabled via the CALDI command.

SDC $01000C_2C_1C_0$

SDC (Set Default Calibration time) sets the default calibration time (Tcal) according to the three bit code embedded in the lowest three bits of the command. The calibration times are shown in Figure 10. Note that the actual duration of a calibration cycle is approximately $2 \times T(cal) + 9$ milliseconds.

WRNV $10011A_2A_1A_0$

WRNV (WRite NonVolatile) writes the user supplied data, in the PARAMETER 1 and PARAMETER 2 registers, into the user accessible area of the AD1170's nonvolatile memory. Eight words of this memory are available, and are addressed by the lowest three bits of the command.

RDNV $10100A_2A_1A_0$

RDNV (ReaD NonVolatile) reads one word from the user accessible portion of the nonvolatile memory within the AD1170, and places the data into the LOW DATA and MID DATA registers for retrieval by the user. The address of the desired word is embedded into the lowest three bits of the command.

RST 10010000

RST (ReSeT) is effectively equivalent to a hardware reset of the AD1170. After executing this command, all nonvolatile parameters (including the ECAL coefficient, the default integration and calibration periods, EIS/ELS period, NULL value and the default format) will be restored to their last saved values, automatic calibration will be enabled, and NULL will be disabled.

NULL 01110000

NULL measures the input signal (using the curent integration time value) and stores the measurement in internal RAM. It allows the user to establish the value of offset voltage at the input and subtract that offset from subsequent conversions through the execution of the NULEN command. The user must insure that the sum of the offset value plus the full scale input is less than the ±6 volts linear input range of the AD1170. Ideally the offset value to be nulled should be no more than a few hundred millivolts in amplitude. The value measured by the NULL command is saved and restored by the SAVA and RESA commands – maintaining this value through subsequent powerups. The NULL command need only be invoked when a new null measurement is desired.

NULEN 01111000

NULEN (NULl ENable) subtracts the value, measured and stored by the last NULL command, from all subsequent conversions. When NULEN is in effect, each conversion's length will be extended by approximately 700 microseconds.

NULDI 10000000

NULDI (NULl DIsable) cancels the effect of the NULEN command.

[1] This logic signal should be a TTL or CMOS compatible continuous waveform. It need not be symmetrical, but the HIGH or LOW time should not be less than 25 microseconds.

IBM PC* INTERFACE

Figure 12 is an example of an AD1170/IBM interface suitable for the IBM PC or XT personal computers. In this case, the AD1170 is interfaced in the I/O space; the DIP switch controls the specific location of the AD1170 within the available address space.

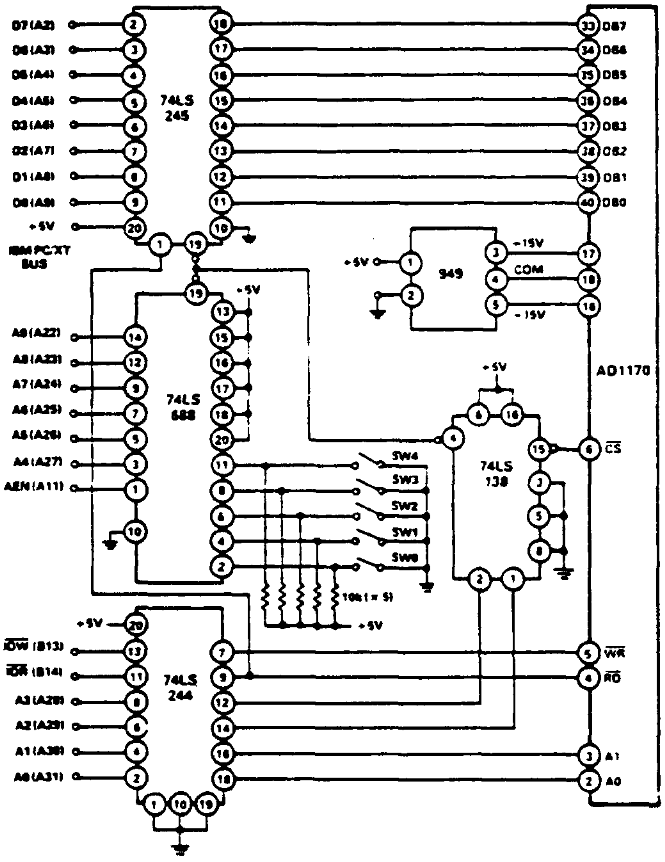


Figure 12. IBM PC/XT to AD1170 Interface

INTERFACING TO AN 8051 MICROCONTROLLER

Figure 13 shows how an AD1170 may be interfaced to an 8051 microcontroller using a technique commonly called "byte banging", where the control lines and data bus of a device are manipulated under firmware control. This "byte banging" technique can be adapted to most microprocessors and is useful in situations where a conventional bus structure is either nonexistent or unavailable for use.[1]

The AD1170's data bus is connected to the 8051 using I/O lines P2.0 through P2.7. The address lines A0 and A1 are connected to I/O lines P1.0 and P1.1 respectively. The RD/ and WR/ lines are connected to P1.2 and P1.3. The CS/ line of the AD1170 is grounded when it is the only device connected to the 8051, but multiple AD1170s could easily be connected in the same way if each CS/ line were separately controlled.

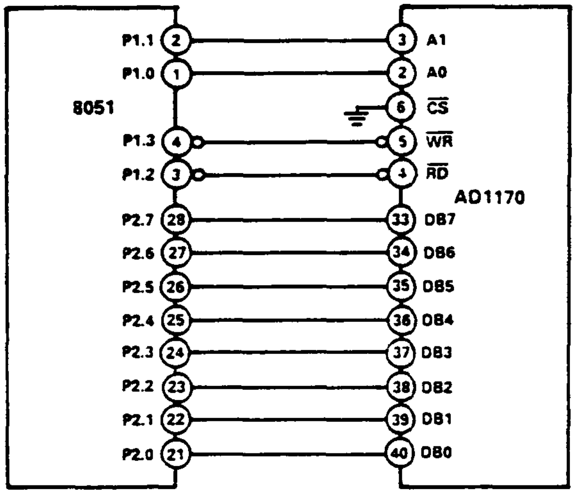


Figure 13. Simple 8051 to AD1170 Interface

To initialize the interface, first write "1"s to the port pins connected to the data bus and the RD/ and WR/ control lines. This puts the 8051 I/O lines into a lightly "pulled up" state, simulating a tri-stated condition on the bus to insure that neither RD/ or WR/ are selected:

```
INIT:     SETB  P1.2          ;DISABLE RD/
          SETB  P1.3          ;AND WR/
                              ;
          MOV   P2, #OFFH     ;SET P2 TO ALL ONES
```

To write a command to the AD1170, first set the state of the P1.1 and P1.0 lines for the address corresponding to the byte to be written to. Set the P2 port to the command data, then strobe the WR/ line by first clearing the P1.3 line and then setting it:

```
WRCMD:    CLR   P1.0          ;FIRST CLEAR A0 AND A1
          CLR   P1.1          ;TO POINT TO CMD BYTE
                              ;
          MOV   P2, #CNV      ;CNV IS THE OPCODE FOR
                              ;A SINGLE CONVERSION
                              ;
          CLR   P1.3          ;STROBE THE WR/ LINE
          SETB  P1.3          ;ONE TIME
                              ;
          MOV   P2, #OFFH     ;CLEAR DATA BUS TO
                              ;ALL ONES
```

To read a byte from the AD1170, first set the P1.0 and P1.1 lines to point to the address of the byte desired. Bring the RD/ line low, reading the contents of P2. Return the RD/ line high:

```
RDSTAT:   CLR   P1.0          ;POINT TO STATUS BYTE
          CLR   P1.1          ;
                              ;
          CLR   P1.2          ;BRING RD LINE LOW
          MOV   A,P2          ;READ CONTENTS OF BUS
          SETB  P1.2          ;RESTORE RD LINE HIGH
```

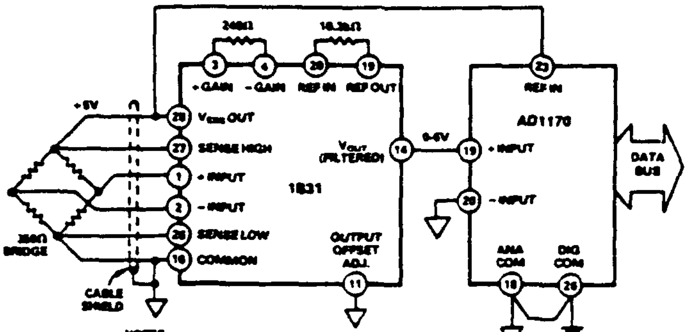
[1] Note that the 8051 microcontroller *does* contain a conventional bus structure; the "byte banging" interface shown here is presented as an example of an alternative technique.

*IBM PC is a trademark of International Business Machines Corp.

:SSURE TRANSDUCER DATA ACQUISITION

ro module solution for microcomputer based data acquisition ; a 1B31 hybrid signal conditioner and an AD1170 as shown igure 14. A 3 millivolt/volt pressure transducer (e.g., Dynisco's series) is interfaced to a model 1B31 configured for a gain 33.3, to provide a 0 to 5 volt output. The regulated excitation age is 5 volts, and is used as the reference input for the 1170 to produce ratiometric operation. This configuration ds very high CMR enhanced by the 1B31 low pass filter and integrating conversion scheme of the AD1170.

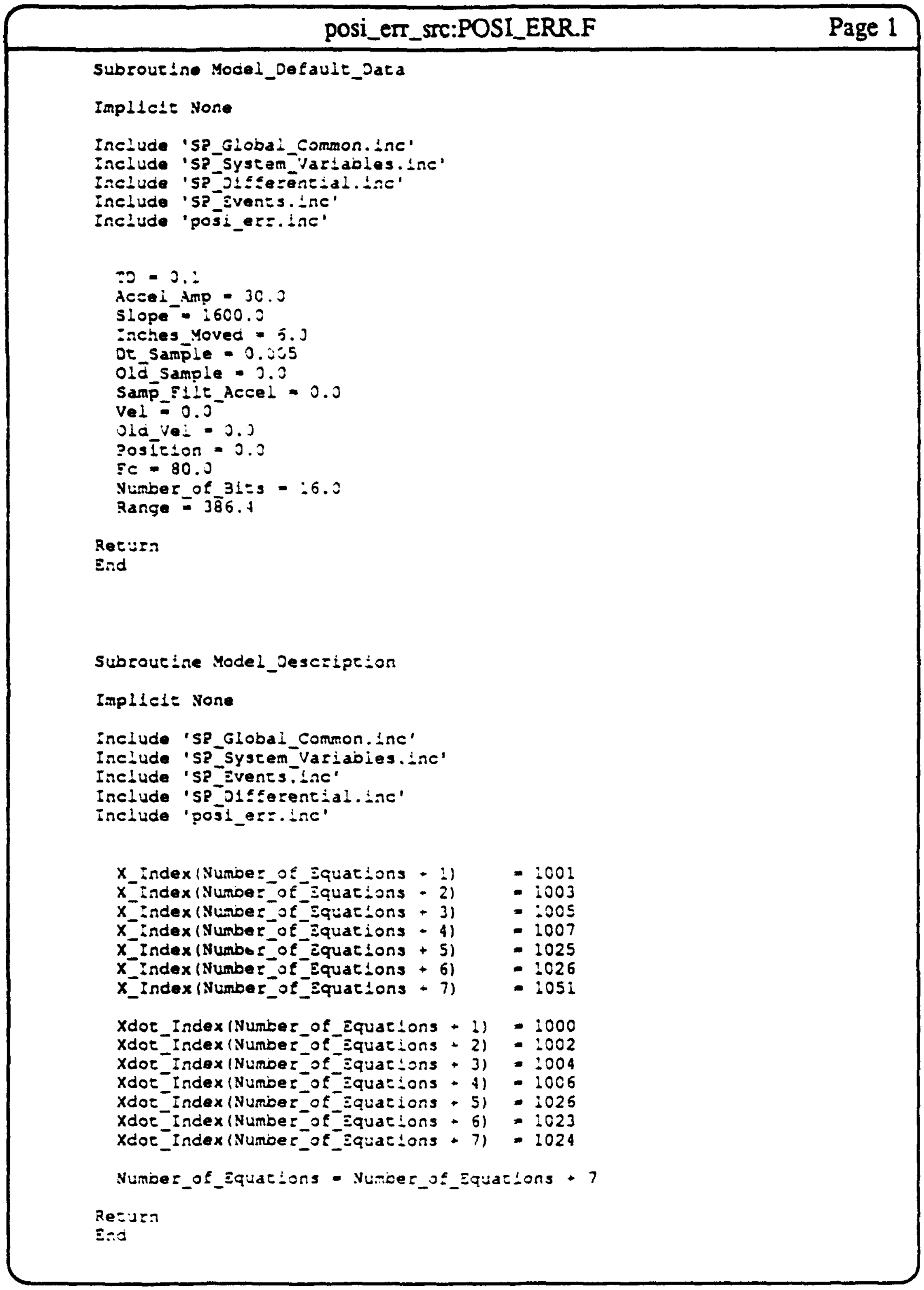
addition, fixed offsets caused by bridge imbalance can be led out by the AD1170 with a power-up initialization command m the microcomputer (see COMPENSATION OF EXTER-L OFFSETS section). The full-scale output of the 1B31 and insducer can also be normalized to AD1170 full scale through electronic calibration command ECAL. Both the offset and l-scale correction data can then be stored in nonvolatile memory eliminate repeating this trim process after each power-up. e AD1170 eliminates a potentiometer or software overhead ich might otherwise be needed for these functions.



NOTES
- 3mV/V LOAD CELL
- +5V EXCITATION, 15mV FS
- GAIN = 333
- USE 10ppm/°C GAIN RESISTOR FOR LOW GAIN TEMPCO

Figure 14. Pressure Transducer Data Acquisition Using 1B31 and AD1170

Appendix B - Simulation Program

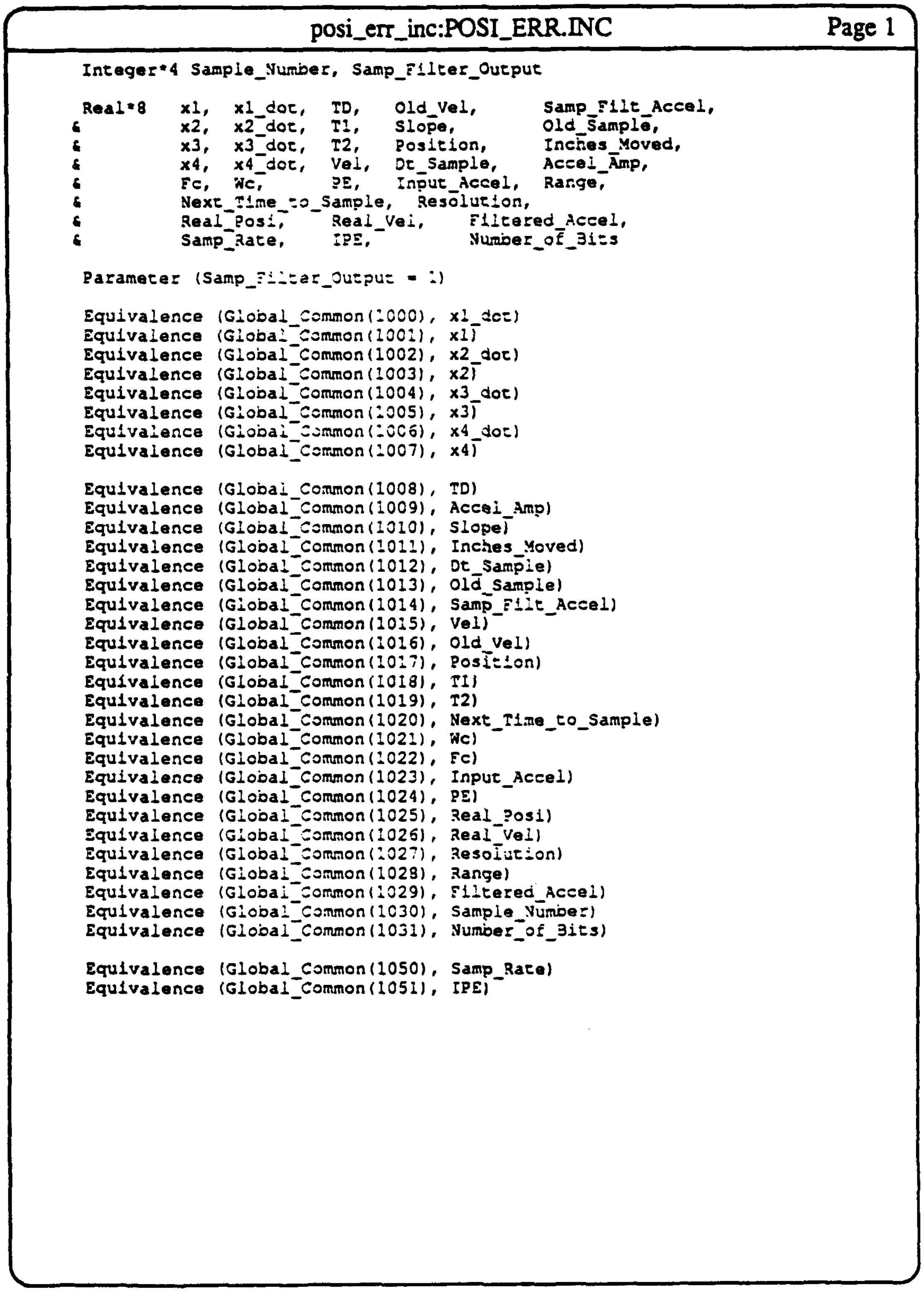


posi_err_src:POSI_ERR.F Page 1

```
      Subroutine Model_Default_Data

      Implicit None

      Include 'SP_Global_Common.inc'
      Include 'SP_System_Variables.inc'
      Include 'SP_Differential.inc'
      Include 'SP_Events.inc'
      Include 'posi_err.inc'


        TD = 0.1
        Accel_Amp = 30.0
        Slope = 1600.0
        Inches_Moved = 5.0
        Dt_Sample = 0.005
        Old_Sample = 0.0
        Samp_Filt_Accel = 0.0
        Vel = 0.0
        Old_Vel = 0.0
        Position = 0.0
        Fc = 80.0
        Number_of_Bits = 16.0
        Range = 386.4

      Return
      End



      Subroutine Model_Description

      Implicit None

      Include 'SP_Global_Common.inc'
      Include 'SP_System_Variables.inc'
      Include 'SP_Events.inc'
      Include 'SP_Differential.inc'
      Include 'posi_err.inc'


        X_Index(Number_of_Equations + 1)      = 1001
        X_Index(Number_of_Equations + 2)      = 1003
        X_Index(Number_of_Equations + 3)      = 1005
        X_Index(Number_of_Equations + 4)      = 1007
        X_Index(Number_of_Equations + 5)      = 1025
        X_Index(Number_of_Equations + 6)      = 1026
        X_Index(Number_of_Equations + 7)      = 1051

        Xdot_Index(Number_of_Equations + 1)   = 1000
        Xdot_Index(Number_of_Equations + 2)   = 1002
        Xdot_Index(Number_of_Equations + 3)   = 1004
        Xdot_Index(Number_of_Equations + 4)   = 1006
        Xdot_Index(Number_of_Equations + 5)   = 1026
        Xdot_Index(Number_of_Equations + 6)   = 1023
        Xdot_Index(Number_of_Equations + 7)   = 1024

        Number_of_Equations = Number_of_Equations + 7

      Return
      End
      Subroutine Setup_Model

      Implicit None

      Include 'SP_Global_Common.inc'
      Include 'SP_System_Variables.inc'
      Include 'SP_Events.inc'
      Include 'SP_Differential.inc'
      Include 'posi_err.inc'

      Real*8 P_total, A_coef, B_coef, C_coef, T2A

        Fc = Samp_rate/4.0

        A_coef = Accel_Amp*(1.0 + Accel_Amp/(2.0*Slope))
        B_coef = 5.0*Accel_Amp**2.0/(2.0*Slope)
        C_coef = 2.0*Accel_Amp**3.0/(Slope**2.0) - Inches_Moved

        T2A = (-B_coef + DSqrt(B_coef**2.0 - 4.0*A_coef*C_coef))/(2.0*A_coef)

        If(T2A .gt. 0.0) Then
          T2 = T2A
        Else
          T2 = (-B_coef - DSqrt(B_coef**2.0 - 4.0*A_coef*C_coef))/(2.0*A_coef)
        End If

        P_total = Accel_Amp*(1.0 + Accel_Amp/(2.0*Slope))*T2**2.0
     &          + T2*(5.0*Accel_Amp**2.0)/(2.0*Slope)
     &          + 2.0*Accel_Amp**3.0/(Slope**2.0)

        Write(*,*) 'P_total = ',P_total

        T1 = Accel_Amp/Slope

        Wc = 2.0*3.1415*Fc

        Resolution = Range/(2.0**Number_of_Bits)
        Write(*,*) 'Resolution = ', Resolution, ' in/sec2  or', Resolution/386.4,
        Write(*,*) 'Bandwidth = ', Fc, ' Hz'
        Dt_Sample = 1.0/Samp_Rate
        Write(*,*) 'Sampling Rate = ', Samp_Rate, ' Hz'
        Write(*,*) 'Normal Simulation Rate = ', 1.0/Dtmax, ' Hz'
        Write(*,*) 'Maximum Simulation Rate = ', 1.0/Dtmin, ' Hz'

        Sample_Number = 1
        Next_Time_to_Sample = Dt_Sample
        Call SP_Schedule(Next_Time_to_Sample, Samp_Filter_Output)

      Return
      End



      Subroutine Continuous_Model

      Implicit None

      Include 'SP_Global_Common.inc'
      Include 'SP_System_Variables.inc'
```

```
        Include 'SP_Events.inc'
        Include 'SP_Differential.inc'
        Include 'posi_err.inc'

        Integer*4 i

        Real*8 Step

          Input_Accel = Slope*(t - Td)*Step(Td)
     &                - Slope*(t - (Td + 2.0*T1))*Step(Td + 2.0*T1)
     &                - Slope*(t - (Td + 2.0*T1 + T2/2.0))*Step(Td + 2.0*T1 + T2/2.
     &                + Slope*(t - (Td + 5.0*T1 + T2/2.0))*Step(Td + 5.0*T1 + T2/2.
     &                + Slope*(t - (Td + 5.0*T1 + 1.5*T2))*Step(Td + 5.0*T1 + 1.5*T
     &                - 2.0*Slope*(t - (Td + 6.0*T1 + 1.5*T2))*Step(Td + 6.0*T1 + 1.
     &                + 2.0*Slope*(t - (Td + 7.0*T1 + 1.5*T2))*Step(Td + 7.0*T1 + 1.
     &                - 2.0*Slope*(t - (Td + 8.0*T1 + 1.5*T2))*Step(Td + 8.0*T1 + 1.
     &                + 2.0*Slope*(t - (Td + 9.0*T1 + 1.5*T2))*Step(Td + 9.0*T1 + 1.
     &                - 2.0*Slope*(t - (Td + 10.0*T1 + 1.5*T2))*Step(Td + 10.0*T1 +
     &                + 2.0*Slope*(t - (Td + 11.0*T1 + 1.5*T2))*Step(Td + 11.0*T1 +
     &                - Slope*(t - (Td + 12.0*T1 + 1.5*T2))*Step(Td + 12.0*T1 + 1.5

c           Input_Accel = Slope*(t - Td)*Step(Td)
c     &                 - Slope*(t - (Td + T1))*Step(Td + T1)
c     &                 - Slope*(t - (Td + T1 + T2))*Step(Td + T1 + T2)
c     &                 + Slope*(t - (Td + 3.0*T1 + T2))*Step(Td + 3.0*T1 + T2)
c     &                 + Slope*(t - (Td + 3.0*T1 + 2.0*T2))*Step(Td + 3.0*T1 + 2.0*T
c     &                 - Slope*(t - (Td + 4.0*T1 + 2.0*T2))*Step(Td + 4.0*T1 + 2.0*T

          x1_dot = -0.7654*Wc*x1 + x2
          x2_dot = -Wc*Wc*x1 + Wc*Wc*Input_Accel
          x3_dot = -1.8478*Wc*x3 + x4
          x4_dot = -Wc*Wc*x3 + Wc*Wc*x1

          Filtered_Accel = x3

          PE = Real_Posi - Position

c          Write(*,*) 'In Continuous_Model at t = ',t
c          Do i = 1000,1029
c            Write(*,*) i,Global_Common(i)
c          End Do
c
c          Do i = 1030,1031
c            Write(*,*) i,Integer_Global_Common(1,i)
c          End Do

        Return
        End



        Subroutine Discrete_Model

        Implicit None

        Include 'SP_Global_Common.inc'
        Include 'SP_System_Variables.inc'
        Include 'SP_Events.inc'
        Include 'SP_Differential.inc'
        Include 'posi_err.inc'

        Integer*4 i
```

```
c          Write(*,*) 'In Discrete_Model at t = ',t
c        Do i = 1000,1029
c           Write(*,*) i,Global_Common(i)
c         End Do
c
c         Do i = 1030,1031
c           Write(*,*) i,Integer_Global_Common(1,i)
c         End Do

        If (Events(1) .eq. Samp_Filter_Output) Then
          Old_Sample = Samp_Filt_Accel
          Samp_Filt_Accel = Resolution*DInt(Filtered_Accel/Resolution)

          Old_Vel = Vel
          Vel = Vel + (Old_Sample + Samp_Filt_Accel)*Dt_Sample/2.0

          Position = Position + (Old_Vel + Vel)*Dt_Sample/2.0

          Sample_Number = Sample_Number + 1
          Next_Time_to_Sample = Dt_Sample*Sample_Number

          Call SP_Schedule(Next_Time_to_Sample, Samp_Filter_Output)

        End If


      Return
      End



      Subroutine Model_Termination

      Implicit None

      Include 'SP_Global_Common.inc'
      Include 'SP_System_Variables.inc'
      Include 'SP_Differential.inc'
      Include 'SP_Events.inc'
      Include 'posi_err.inc'

        If (t .ge. (Td + 12.0*T1 + 1.5*T2 + 0.1)) Then
          End_of_Run = .TRUE.
        End If

      Return
      End
      Double Precision Function Step(Time to step)

*----------------------------------------------------------------------+
*  This function is the step function.                                 |
*  i.e. Step(Time to step) = 1.0   if t >= Time to step            |
*       Step(Time to step) = 0.0   if t <  Time to step            |
*----------------------------------------------------------------------+

*----------------------------------------------------------------------+
*  Include global common block definitions                             |
*----------------------------------------------------------------------+

      Implicit None

      Include 'SP_Global_Common.inc'
      Include 'SP_System_Variables.inc'

*----------------------------------------------------------------------+
*  Variable Declarations                                               |
*----------------------------------------------------------------------+

      Double Precision Time to step

*----------------------------------------------------------------------+
*  Begin Code                                                          |
*----------------------------------------------------------------------+

      If (t .ge. Time to step) Then
        Step = 1.0d0
      Else
        Step = 0.0d0
      End If

      Return
      End
```



posi_err_inc:POSI_ERR.INC Page 1

```
 Integer*4 Sample_Number, Samp_Filter_Output

 Real*8    x1,  x1_dot,  TD,   Old_Vel,      Samp_Filt_Accel,
&          x2,  x2_dot,  T1,   Slope,        Old_Sample,
&          x3,  x3_dot,  T2,   Position,     Inches_Moved,
&          x4,  x4_dot,  Vel,  Dt_Sample,    Accel_Amp,
&          Fc,  Wc,      PE,   Input_Accel,  Range,
&          Next_Time_to_Sample,  Resolution,
&          Real_Posi,    Real_Vel,    Filtered_Accel,
&          Samp_Rate,    IPE,         Number_of_Bits

 Parameter (Samp_Filter_Output = 1)

 Equivalence (Global_Common(1000), x1_dot)
 Equivalence (Global_Common(1001), x1)
 Equivalence (Global_Common(1002), x2_dot)
 Equivalence (Global_Common(1003), x2)
 Equivalence (Global_Common(1004), x3_dot)
 Equivalence (Global_Common(1005), x3)
 Equivalence (Global_Common(1006), x4_dot)
 Equivalence (Global_Common(1007), x4)

 Equivalence (Global_Common(1008), TD)
 Equivalence (Global_Common(1009), Accel_Amp)
 Equivalence (Global_Common(1010), Slope)
 Equivalence (Global_Common(1011), Inches_Moved)
 Equivalence (Global_Common(1012), Dt_Sample)
 Equivalence (Global_Common(1013), Old_Sample)
 Equivalence (Global_Common(1014), Samp_Filt_Accel)
 Equivalence (Global_Common(1015), Vel)
 Equivalence (Global_Common(1016), Old_Vel)
 Equivalence (Global_Common(1017), Position)
 Equivalence (Global_Common(1018), T1)
 Equivalence (Global_Common(1019), T2)
 Equivalence (Global_Common(1020), Next_Time_to_Sample)
 Equivalence (Global_Common(1021), Wc)
 Equivalence (Global_Common(1022), Fc)
 Equivalence (Global_Common(1023), Input_Accel)
 Equivalence (Global_Common(1024), PE)
 Equivalence (Global_Common(1025), Real_Posi)
 Equivalence (Global_Common(1026), Real_Vel)
 Equivalence (Global_Common(1027), Resolution)
 Equivalence (Global_Common(1028), Range)
 Equivalence (Global_Common(1029), Filtered_Accel)
 Equivalence (Global_Common(1030), Sample_Number)
 Equivalence (Global_Common(1031), Number_of_Bits)

 Equivalence (Global_Common(1050), Samp_Rate)
 Equivalence (Global_Common(1051), IPE)
```

Appendix C - Real-Time Position Determination Program

```
#include <graph.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <bios.h>
#include "iarinc.h"

void
    calibrate_AD1170 (void),
    calibrate_QA2000 (void),
    command_anorad (char *ano_buffer),
    display_menu_item (int left_row, int left_col, char *str, int len),
    draw_menu_box (int up_left_row, int up_left_col, int height, int width),
    free_run_mode (void),
    setup_ADC_defaults (void),
    setup_anorad (void),
    main (void),
    wait_for (double seconds),
    wait_for_A (void),
    wait_for_B (void);

int
    get_menu (int row, int col, char * *items);

unsigned
    cursor (unsigned value);

double
    do_zupt (int numb_data_points),
    exper_1 (void),
    exper_2 (void),
    exper_3 (void),
    exper_4 (void),
    exper_5 (void),
    exper_6 (void),
    integrate_and_move (double seconds, char *ano_buffer),
    read_A (void),
    read_B (void),
    read_position (void),
    setup_ADC_int_time (double frequency),
    setup_counter (double frequency);

/* Main Menu Definitions */

char
    *mnuMain[] =
      {
        "System Setup",
        "Experiments",
        "Free-Running Position Display",
        "Quit",
        NULL
      };

#define SYS_SETUP 0
#define EXPER     1
#define FREE_RUN  2
#define QUIT      3

/* Menu for System Setup */
```

```
char
    *mnuSys[] =
      {
        "Calibrate QA2000",
        "Calibrate AD1170's",
        "Select Integration Method",
        "Select Sampling Rate",
        "RETURN TO MAIN MENU",
        NULL
      };

#define CAL_QA2000       0
#define CAL_AD1170       1
#define SEL_INTEG        2
#define SEL_SAMP         3
#define RETURN_FROM_SYS  4

/* Menu for Experiments */

char
    *mnuExp[] =
      {
        "Single Motion Test",
        "Multiple Motion Test w/ZUPT",
        "Multiple Motion Test w/ZUPT & Unkown Base",
        "Multiple Motion Test w/ZUPT & Known Base",
     "Multiple Motion Test w/ZUPT & 2 Known Bases (M1)",
     "Multiple Motion Test w/ZUPT & 2 Known Bases (M2)",
        "RETURN TO MAIN MENU",
        NULL
      };

#define SINGLE            0
#define ZUPT              1
#define ZUPT_BASE         2
#define ZUPT_KNOWN_BASE       3
#define ZUPT_2_KNOWN_BASES    4
#define ZUPT_2_KNOWN_BASES_M2 5
#define RETURN_FROM_EXP       6

/* Menu for Integration Methods */

char
    *mnuInteg[] =
      {
        "Simpson's Rule",
        "Trapezoidal",
        "Adam-Basforth",
        "RETURN TO MAIN MENU",
        NULL
      };

#define SIMP              0
#define TRAP              1
#define ADAM              2
#define RETURN_FROM_INTEG 3

int
    integ_alg, center_row, center_col;

double
    SF,  bias, bias_time,  vel,  dt,  dt_2,  dt_12,  dt_24;
```

```
/* Video configuration structure (graph.h) */

struct videoconfig
    vid_config;

/* Structure for menu attributes */

struct mnuAtr
  {
    int     fgNormal, fgSelect, fgBorder;
    long    bgNormal, bgSelect, bgBorder;
    int     centered;
    char    nw[2], ne[2], se[2], sw[2], ns[2], ew[2];
  };

/* Color menu attributes */

struct mnuAtr
    menus =
      {
        0x0f, 0x01, 0x0f,
        0x01, 0x0f, 0x01,
        TRUE,
        "┌", "┐", "┘", "└", "│", "─"
      };

/* Monochrome menu attributes */

struct mnuAtr
    bwmenus =
      {
        0x07, 0x00, 0x07,
        0x00, 0x07, 0x00,
        TRUE,
        "┌", "┐", "┘", "└", "│", "─"
      };


FILE
    *log_ptr;



void main ()
  {
    char
     string[80],  file_name[80];

    int
     choice,  i_cnt,  numb_runs;

    double
     samp_freq,  freq,  int_freq,  p_err,  p_err_sum,  p_err_avg,
     p_err_sq_sum,  p_err_sq_avg,  max_err,  f_band,  ad_range;

 /*****************************************************************/
 /*  Open output file                                        */
 /*****************************************************************/

  printf ("\nEnter name of log file: ");
  gets (file_name);
```

```
log_ptr = fopen (file_name, "w");
if (log_ptr == NULL)
 {
  printf ("\n\n\t***  Could not open %s for writing  ***", file_name);
  exit(0);
 }


  setup_anorad ();

  outp (CNTR_STOP, 0x0000);            /* Disarm 2MCLK  */

  setup_ADC_defaults ();

  freq = 200.0;
  integ_alg = TRAP;
  bias = 0.0;

  samp_freq = setup_counter (freq);

  dt = 1/samp_freq;
  dt_2 = dt/2.0;
  dt_12 = dt/12.0;
  dt_24 = dt/24.0;

  int_freq = setup_ADC_int_time (samp_freq);

  outp (CNTR_START, 0x0000);            /* Arm 2MCLK  */

  _getvideoconfig (&vid_config);
  center_row = vid_config.numtextrows / 2;
  center_col = vid_config.numtextcols / 2;

  _setvideomode (_DEFAULTMODE);

  calibrate_QA2000 ();

  _setbkcolor ((long) 0);
  _clearscreen (_GCLEARSCREEN);
  _setbkcolor ((long) 1);
  _settextcolor ((short) 15);
  _settextposition (center_row, center_col - 13);
  _outtext ("Enter filter bandwidth (Hz): ");
  gets (string);
  f_band = atof (string);

  _setbkcolor ((long) 0);
  _clearscreen (_GCLEARSCREEN);
  _setbkcolor ((long) 1);
  _settextcolor ((short) 15);
  _settextposition (center_row, center_col - 13);
  _outtext ("Enter A-to-D range (Gs): ");
  gets (string);
  ad_range = atof (string);

  _setbkcolor ((long) 0);
  _clearscreen (_GCLEARSCREEN);

  /* Select and branch to menu choices */

  do
    {
        _setbkcolor ((long) 1);
        _settextcolor ((short) 15);
     _settextposition (1, center_col - 13);
        _outtext (" IAR DEMONSTRATION PROGRAM ");

        choice = get_menu (center_row, center_col, mnuMain);

        switch (choice)
          {
            case SYS_SETUP:
              choice = get_menu (center_row, center_col, mnuSys);
              switch (choice)
                {
                  case CAL_QA2000:
                    calibrate_QA2000 ();
                  break;

                  case CAL_AD1170:
                    calibrate_AD1170 ();
                  break;

                  case SEL_INTEG:
                    choice = get_menu (center_row, center_col, mnuInteg);
                    switch (choice)
                      {
                        case SIMP:
                          integ_alg = SIMP;
                        break;

                        case TRAP:
                          integ_alg = TRAP;
                        break;

                        case ADAM:
                          integ_alg = ADAM;
                        break;
                      }
                  break;

                  case SEL_SAMP:
                      outp (CNTR_STOP, 0x0000);                 /* Disarm 2MCLK
*/

                    do
                      {
               _setbkcolor ((long) 0);
                        _clearscreen (_GCLEARSCREEN);
                        _setbkcolor ((long) 1);
                        _settextcolor ((short) 15);
                        _settextposition (center_row, center_col - 13);
                        _outtext ("Enter sampling rate (Hz): ");
                        gets (string);
                        freq = atof (string);
                }
                    while (freq < 31.0 || freq > 490.0);

                    samp_freq = setup_counter (freq);

                    dt = 1/samp_freq;
                    dt_2 = dt/2.0;
                    dt_12 = dt/12.0;
                    dt_24 = dt/24.0;
        int_freq = setup_ADC_int_time (samp_freq);

        outp (CNTR_START, 0x0000);           /* Arm 2MCLK  */

      break;

      case RETURN_FROM_SYS:
      break;
    }
break;

case EXPER:
  choice = get_menu (center_row, center_col, mnuExp);
  switch (choice)
    {
case SINGLE:

  fprintf (log_ptr, "\n\n\n\n\t\t\tSINGLE\n");

  _setbkcolor ((long) 0);
  _clearscreen (_GCLEARSCREEN);
  _setbkcolor ((long) 1);
  _settextcolor ((short) 15);
  _settextposition (center_row, center_col - 11);
  _outtext ("Enter number of runs: ");
  gets (string);
  numb_runs = atoi (string);

  bias_time = 0.0;

  p_err = 0.0;
  p_err_sum = 0.0;
  p_err_sq_sum = 0.0;
  max_err = 0.0;

  for (i_cnt = 0; i_cnt < numb_runs; ++i_cnt)
    {
   p_err = exper_1 ();
   p_err_sum = p_err_sum + p_err;
   p_err_sq_sum = p_err_sq_sum + p_err*p_err;

   if (fabs(max_err) < fabs(p_err))
     {
       max_err = p_err;
     }
    }

  p_err_avg = p_err_sum/((double) numb_runs);
  p_err_sq_avg = p_err_sq_sum/((double) numb_runs);

  _clearscreen (_GCLEARSCREEN);
  _settextposition (1, center_col - 3);
  _outtext ("SINGLE");

  _settextposition (3, 1);

  sprintf (string,
        "Mean Error = %12.4g (inches)    STD = %12.4g (inches)",
        p_err_avg,
        sqrt(p_err_sq_avg - p_err_avg*p_err_avg));
  _outtext (string);

  sprintf (string, "\nMax Error = %12.4g (inches)", max_err);
```

```
              _outtext (string);

              sprintf (string, "\nSampling rate = %g (Hz)", samp_freq);
              _outtext (string);

              sprintf (string, "\nFilter bandwidth = %g (Hz)", f_band);
              _outtext (string);

              sprintf (string, "\nA-to-D Range = %g (Gs)", ad_range);
              _outtext (string);

              sprintf (string, "\nIntegration method = %d ", integ_alg);
              _outtext (string);

              _outtext ("\n      (0-Simp  1-Trap  2-Adam)");

              sprintf (string, "\nNumber of runs = %d ", numb_runs);
              _outtext (string);

              fprintf (log_ptr,
                      "\n\nMean Error = %12.4g (inches)      STD = %12.4g
(inches)",
                    p_err_avg,
                    sqrt(p_err_sq_avg - p_err_avg*p_err_avg));
              fprintf (log_ptr, "\nMax Error = %12.4g (inches)", max_err);
              fprintf (log_ptr, "\nSampling rate = %g (Hz)", samp_freq);
              fprintf (log_ptr, "\nFilter bandwidth = %g (Hz)", f_band);
              fprintf (log_ptr, "\nA-to-D Range = %g (Gs)", ad_range);
              fprintf (log_ptr, "\nIntegration method = %d ", integ_alg);
              fprintf (log_ptr, "\n      (0-Simp  1-Trap  2-Adam)");
              fprintf (log_ptr, "\nNumber of runs = %d ", numb_runs);

              _settextposition (center_row+2, center_col - 19);
              _outtext ("Press <Enter> to return to main menu.");
              gets (string);

                  break;


            case ZUPT:

              fprintf (log_ptr, "\n\n\n\n\t\t\tZUPT ONLY\n");

              _setbkcolor ((long) 0);
              _clearscreen (_GCLEARSCREEN);
              _setbkcolor ((long) 1);
              _settextcolor ((short) 15);
              _settextposition (center_row, center_col - 11);
              _outtext ("Enter number of runs: ");
              gets (string);
              numb_runs = atoi (string);

              bias_time = 0.0;

              p_err = 0.0;
              p_err_sum = 0.0;
              p_err_sq_sum = 0.0;
              max_err = 0.0;

              for (i_cnt = 0; i_cnt < numb_runs; ++i_cnt)
                {
               p_err = exper_2 ();
               p_err_sum = p_err_sum + p_err;
               p_err_sq_sum = p_err_sq_sum + p_err*p_err;

               if (fabs(max_err) < fabs(p_err))
                 {
                   max_err = p_err;
                 }
                }

              p_err_avg = p_err_sum/((double) numb_runs);
              p_err_sq_avg = p_err_sq_sum/((double) numb_runs);

              _clearscreen (_GCLEARSCREEN);
              _settextposition (1, center_col - 4);
              _outtext ("ZUPT ONLY");

              _settextposition (3, 1);

              sprintf (string,
                    "Mean Error = %12.4g (inches)    STD = %12.4g (inches)",
                    p_err_avg,
                    sqrt(p_err_sq_avg - p_err_avg*p_err_avg));
              _outtext (string);

              sprintf (string, "\nMax Error = %12.4g (inches)", max_err);
              _outtext (string);

              sprintf (string, "\nSampling rate = %g (Hz)", samp_freq);
              _outtext (string);

              sprintf (string, "\nFilter bandwidth = %g (Hz)", f_band);
              _outtext (string);

              sprintf (string, "\nA-to-D Range = %g (Gs)", ad_range);
              _outtext (string);

              sprintf (string, "\nIntegration method = %d ", integ_alg);
              _outtext (string);

              _outtext ("\n      (0-Simp  1-Trap  2-Adam)");

              sprintf (string, "\nNumber of runs = %d ", numb_runs);
              _outtext (string);

              fprintf (log_ptr,
                       "\n\nMean  Error  =  %12.4g  (inches)      STD  =  %12.4g
(inches)",
                    p_err_avg,
                    sqrt(p_err_sq_avg - p_err_avg*p_err_avg));
              fprintf (log_ptr, "\nMax Error = %12.4g (inches)", max_err);
              fprintf (log_ptr, "\nSampling rate = %g (Hz)", samp_freq);
              fprintf (log_ptr, "\nFilter bandwidth = %g (Hz)", f_band);
              fprintf (log_ptr, "\nA-to-D Range = %g (Gs)", ad_range);
              fprintf (log_ptr, "\nIntegration method = %d ", integ_alg);
              fprintf (log_ptr, "\n      (0-Simp  1-Trap  2-Adam)");
              fprintf (log_ptr, "\nNumber of runs = %d ", numb_runs);

              _settextposition (center_row+2, center_col - 19);
              _outtext ("Press <Enter> to return to main menu.");
              gets (string);

            break;
```

```
case ZUPT_BASE:

  fprintf (log_ptr, "\n\n\n\n\t\t\tZUPT W/ONE UNKNOWN BASE\n");

  _setbkcolor ((long) 0);
  _clearscreen (_GCLEARSCREEN);
  _setbkcolor ((long) 1);
  _settextcolor ((short) 15);
  _settextposition (center_row, center_col - 11);
  _outtext ("Enter number of runs: ");
  gets (string);
  numb_runs = atoi (string);

  p_err = 0.0;
  p_err_sum = 0.0;
  p_err_sq_sum = 0.0;
  max_err = 0.0;

  for (i_cnt = 0; i_cnt < numb_runs; ++i_cnt)
    {
   p_err = exper_3 ();
   p_err_sum = p_err_sum + p_err;
   p_err_sq_sum = p_err_sq_sum + p_err*p_err;

   if (fabs(max_err) < fabs(p_err))
     {
       max_err = p_err;
     }
    }

  p_err_avg = p_err_sum/((double) numb_runs);
  p_err_sq_avg = p_err_sq_sum/((double) numb_runs);

  _clearscreen (_GCLEARSCREEN);
  _settextposition (1, center_col - 11);
  _outtext ("ZUPT W/ONE UNKOWN BASE");

  _settextposition (3, 1);

  sprintf (string,
        "Mean Error = %12.4g (inches)    STD = %12.4g (inches)",
        p_err_avg,
        sqrt(p_err_sq_avg - p_err_avg*p_err_avg));
  _outtext (string);

  sprintf (string, "\nMax Error = %12.4g (inches)", max_err);
  _outtext (string);

  sprintf (string, "\nSampling rate = %g (Hz)", samp_freq);
  _outtext (string);

  sprintf (string, "\nFilter bandwidth = %g (Hz)", f_band);
  _outtext (string);

  sprintf (string, "\nA-to-D Range = %g (Gs)", ad_range);
  _outtext (string);

  sprintf (string, "\nIntegration method = %d ", integ_alg);
  _outtext (string);

  _outtext ("\n      (0-Simp  1-Trap  2-Adam)");

  sprintf (string, "\nNumber of runs = %d ", numb_runs);
```

```
              _outtext (string);

              fprintf (log_ptr,
                       "\n\nMean  Error  =  %12.4g  (inches)      STD  =  %12.4g
(inches)",
                    p_err_avg,
                    sqrt(p_err_sq_avg - p_err_avg*p_err_avg));
              fprintf (log_ptr, "\nMax Error = %12.4g (inches)", max_err);
              fprintf (log_ptr, "\nSampling rate = %g (Hz)", samp_freq);
              fprintf (log_ptr, "\nFilter bandwidth = %g (Hz)", f_band);
              fprintf (log_ptr, "\nA-to-D Range = %g (Gs)", ad_range);
              fprintf (log_ptr, "\nIntegration method = %d ", integ_alg);
              fprintf (log_ptr, "\n      (0-Simp  1-Trap  2-Adam)");
              fprintf (log_ptr, "\nNumber of runs = %d ", numb_runs);

              _settextposition (center_row+2, center_col - 19);
              _outtext ("Press <Enter> to return to main menu.");
              gets (string);

            break;


            case ZUPT_KNOWN_BASE:

              fprintf (log_ptr, "\n\n\n\n\t\t\tZUPT W/ONE BASE\n");

              _setbkcolor ((long) 0);
              _clearscreen (_GCLEARSCREEN);
              _setbkcolor ((long) 1);
              _settextcolor ((short) 15);
              _settextposition (center_row, center_col - 11);
              _outtext ("Enter number of runs: ");
              gets (string);
              numb_runs = atoi (string);

              p_err = 0.0;
              p_err_sum = 0.0;
              p_err_sq_sum = 0.0;
              max_err = 0.0;

              for (i_cnt = 0; i_cnt < numb_runs; ++i_cnt)
                {
               p_err = exper_4 ();
               p_err_sum = p_err_sum + p_err;
               p_err_sq_sum = p_err_sq_sum + p_err*p_err;

               if (fabs(max_err) < fabs(p_err))
                 {
                   max_err = p_err;
                 }
                }

              p_err_avg = p_err_sum/((double) numb_runs);
              p_err_sq_avg = p_err_sq_sum/((double) numb_runs);

              _clearscreen (_GCLEARSCREEN);
              _settextposition (1, center_col - 7);
              _outtext ("ZUPT W/ONE BASE");

              _settextposition (3, 1);

              sprintf (string,
                    "Mean Error = %12.4g (inches)    STD = %12.4g (inches)",
                    p_err_avg,
                    sqrt(p_err_sq_avg - p_err_avg*p_err_avg));
              _outtext (string);

              sprintf (string, "\nMax Error = %12.4g (inches)", max_err);
              _outtext (string);

              sprintf (string, "\nSampling rate = %g (Hz)", samp_freq);
              _outtext (string);

              sprintf (string, "\nFilter bandwidth = %g (Hz)", f_band);
              _outtext (string);

              sprintf (string, "\nA-to-D Range = %g (Gs)", ad_range);
              _outtext (string);

              sprintf (string, "\nIntegration method = %d ", integ_alg);
              _outtext (string);

              _outtext ("\n      (0-Simp  1-Trap  2-Adam)");

              sprintf (string, "\nNumber of runs = %d ", numb_runs);
              _outtext (string);

              fprintf (log_ptr,
                      "\n\nMean  Error  =  %12.4g  (inches)      STD  =  %12.4g
(inches)",
                    p_err_avg,
                    sqrt(p_err_sq_avg - p_err_avg*p_err_avg));
              fprintf (log_ptr, "\nMax Error = %12.4g (inches)", max_err);
              fprintf (log_ptr, "\nSampling rate = %g (Hz)", samp_freq);
              fprintf (log_ptr, "\nFilter bandwidth = %g (Hz)", f_band);
              fprintf (log_ptr, "\nA-to-D Range = %g (Gs)", ad_range);
              fprintf (log_ptr, "\nIntegration method = %d ", integ_alg);
              fprintf (log_ptr, "\n      (0-Simp  1-Trap  2-Adam)");
              fprintf (log_ptr, "\nNumber of runs = %d ", numb_runs);

              _settextposition (center_row+2, center_col - 19);
              _outtext ("Press <Enter> to return to main menu.");
              gets (string);

            break;


            case ZUPT_2_KNOWN_BASES:

              fprintf (log_ptr, "\n\n\n\n\t\t\tZUPT W/2 BASES (M1)\n");

              _setbkcolor ((long) 0);
              _clearscreen (_GCLEARSCREEN);
              _setbkcolor ((long) 1);
              _settextcolor ((short) 15);
              _settextposition (center_row, center_col - 11);
              _outtext ("Enter number of runs: ");
              gets (string);
              numb_runs = atoi (string);

              p_err = 0.0;
              p_err_sum = 0.0;
              p_err_sq_sum = 0.0;
              max_err = 0.0;

              for (i_cnt = 0; i_cnt < numb_runs; ++i_cnt)
```

```
  {
 p_err = exper_5 ();
 p_err_sum = p_err_sum + p_err;
 p_err_sq_sum = p_err_sq_sum + p_err*p_err;

 if (fabs(max_err) < fabs(p_err))
   {
    max_err = p_err;
   }
  }

p_err_avg = p_err_sum/((double) numb_runs);
p_err_sq_avg = p_err_sq_sum/((double) numb_runs);

_clearscreen (_GCLEARSCREEN);
_settextposition (1, center_col - 10);
_outtext ("ZUPT W/2 BASES (M1)");

_settextposition (3, 1);

sprintf (string,
      "Mean Error = %12.4g (inches)    STD = %12.4g (inches)",
      p_err_avg,
      sqrt(p_err_sq_avg - p_err_avg*p_err_avg));
_outtext (string);

sprintf (string, "\nMax Error = %12.4g (inches)", max_err);
_outtext (string);

sprintf (string, "\nSampling rate = %g (Hz)", samp_freq);
_outtext (string);

sprintf (string, "\nFilter bandwidth = %g (Hz)", f_band);
_outtext (string);

sprintf (string, "\nA-to-D Range = %g (Gs)", ad_range);
_outtext (string);

sprintf (string, "\nIntegration method = %d ", integ_alg);
_outtext (string);

_outtext ("\n      (0-Simp  1-Trap  2-Adam)");

sprintf (string, "\nNumber of runs = %d ", numb_runs);
_outtext (string);

fprintf (log_ptr,
        "\n\nMean  Error  =  %12.4g  (inches)      STD  =  %12.4g
(inches)",
      p_err_avg,
      sqrt(p_err_sq_avg - p_err_avg*p_err_avg));
fprintf (log_ptr, "\nMax Error = %12.4g (inches)", max_err);
fprintf (log_ptr, "\nSampling rate = %g (Hz)", samp_freq);
fprintf (log_ptr, "\nFilter bandwidth = %g (Hz)", f_band);
fprintf (log_ptr, "\nA-to-D Range = %g (Gs)", ad_range);
fprintf (log_ptr, "\nIntegration method = %d ", integ_alg);
fprintf (log_ptr, "\n      (0-Simp  1-Trap  2-Adam)");
fprintf (log_ptr, "\nNumber of runs = %d ", numb_runs);

_settextposition (center_row+2, center_col - 19);
_outtext ("Press <Enter> to return to main menu.");
gets (string);
```

```
break;

case ZUPT_2_KNOWN_BASES_M2:

  fprintf (log_ptr, "\n\n\n\n\t\t\tZUPT W/2 BASES (M2)\n");

  _setbkcolor ((long) 0);
  _clearscreen (_GCLEARSCREEN);
  _setbkcolor ((long) 1);
  _settextcolor ((short) 15);
  _settextposition (center_row, center_col - 11);
  _outtext ("Enter number of runs: ");
  gets (string);
  numb_runs = atoi (string);

  p_err = 0.0;
  p_err_sum = 0.0;
  p_err_sq_sum = 0.0;
  max_err = 0.0;

  for (i_cnt = 0; i_cnt < numb_runs; ++i_cnt)
    {
   p_err = exper_6 ();
   p_err_sum = p_err_sum + p_err;
   p_err_sq_sum = p_err_sq_sum + p_err*p_err;

   if (fabs(max_err) < fabs(p_err))
     {
       max_err = p_err;
     }
    }

  p_err_avg = p_err_sum/((double) numb_runs);
  p_err_sq_avg = p_err_sq_sum/((double) numb_runs);

  _clearscreen (_GCLEARSCREEN);
  _settextposition (1, center_col - 10);
  _outtext ("ZUPT W/2 BASES (M2)");

  _settextposition (3, 1);

  sprintf (string,
        "Mean Error = %12.4g (inches)    STD = %12.4g (inches)",
        p_err_avg,
        sqrt(p_err_sq_avg - p_err_avg*p_err_avg));
  _outtext (string);

  sprintf (string, "\nMax Error = %12.4g (inches)", max_err);
  _outtext (string);

  sprintf (string, "\nSampling rate = %g (Hz)", samp_freq);
  _outtext (string);

  sprintf (string, "\nFilter bandwidth = %g (Hz)", f_band);
  _outtext (string);

  sprintf (string, "\nA-to-D Range = %g (Gs)", ad_range);
  _outtext (string);

  sprintf (string, "\nIntegration method = %d ", integ_alg);
  _outtext (string);
```

```
              _outtext ("\n      (0-Simp  1-Trap  2-Adam)");

              sprintf (string, "\nNumber of runs = %d ", numb_runs);
              _outtext (string);

              fprintf (log_ptr,
                      "\n\nMean  Error  =  %12.4g  (inches)      STD  =  %12.4g
(inches)",
                    p_err_avg,
                    sqrt(p_err_sq_avg - p_err_avg*p_err_avg));
              fprintf (log_ptr, "\nMax Error = %12.4g (inches)", max_err);
              fprintf (log_ptr, "\nSampling rate = %g (Hz)", samp_freq);
              fprintf (log_ptr, "\nFilter bandwidth = %g (Hz)", f_band);
              fprintf (log_ptr, "\nA-to-D Range = %g (Gs)", ad_range);
              fprintf (log_ptr, "\nIntegration method = %d ", integ_alg);
              fprintf (log_ptr, "\n      (0-Simp  1-Trap  2-Adam)");
              fprintf (log_ptr, "\nNumber of runs = %d ", numb_runs);

              _settextposition (center_row+2, center_col - 19);
              _outtext ("Press <Enter> to return to main menu.");
              gets (string);

            break;


                  case RETURN_FROM_EXP:
                  break;
                }
         break;

            case FREE_RUN:
              free_run_mode ();
            break;

            case QUIT :
              _setvideomode (_DEFAULTMODE);
              exit(0);
            break;
          }
     _setbkcolor ((long) 0);
        _clearscreen (_GCLEARSCREEN);
      }
    while (1);
  }



void calibrate_AD1170 (void)
  {
    char
        string[80];

    int
     k,  delay_cnt;

    double
        adc_double_a,    adc_double_b;

    outp (CNTR_STOP, 0x0000);

    _setbkcolor ((long) 1);
    _settextcolor ((short) 15);
```

```
 _clearscreen (_GCLEARSCREEN);

 _settextposition (1, 1);
 _outtext ("Apply +5 volt reference to EXT INPUT.");

 _settextposition (2, 1);
 _outtext ("Press <Enter> when ready.");

 gets (string);

/***************************************************************/
/*  Setup AC5004                                               */
/***************************************************************/

 wait_for_A ();
 outp (COMMAND_REG_A, RST);

 wait_for_A ();
 outp (COMMAND_REG_A, SDI | INTEG_167_M);

 wait_for_A ();
 outp (COMMAND_REG_A, SDC | INTEG_300_M);

 wait_for_A ();
 outp (PARAM_1_REG_A, 0x000f);
 outp (COMMAND_REG_A, SDF);

 wait_for_A ();
 outp (COMMAND_REG_A, NULDI);

 wait_for_B ();
 outp (COMMAND_REG_B, RST);

 wait_for_B ();
 outp (COMMAND_REG_B, SDI | INTEG_167_M);

 wait_for_B ();
 outp (COMMAND_REG_B, SDC | INTEG_300_M);

 wait_for_B ();
 outp (PARAM_1_REG_B, 0x000f);
 outp (COMMAND_REG_B, SDF);

 wait_for_B ();
 outp (COMMAND_REG_B, NULDI);

 wait_for_A ();
 outp (COMMAND_REG_A, ECAL);

 wait_for_B ();
 outp (COMMAND_REG_B, ECAL);

 wait_for_A ();
 outp (COMMAND_REG_A, SCAL);

 wait_for_B ();
 outp (COMMAND_REG_B, SCAL);

 wait_for_A ();
 wait_for_B ();

 _settextposition (4, 1);
 _outtext ("Now adjust EXT INPUT to ¯2 volts.");
```

```
    _settextposition (5, 1);
    _outtext ("Press <Enter> when ready.");

    gets (string);

    _clearscreen (_GCLEARSCREEN);

    _settextposition (1, 12);
    _outtext (" Results of Calibration ");

    for (k = 0; k < 10; ++k)
      {
        wait_for_A ();
        outp (COMMAND_REG_A, CNV);

        wait_for_B ();
        outp (COMMAND_REG_B, CNV);

        adc_double_a = read_A ();
        adc_double_b = read_B ();

         _settextposition (2 + k, 1);
     sprintf (string, "\nADC#1 = %13.6g  ADC#2 = %13.6g  Diff = %13.6g",
                  adc_double_a, adc_double_b, adc_double_a - adc_double_b);
     _outtext (string);

     for (delay_cnt = 0; delay_cnt < 30000; ++delay_cnt);
      }

    _settextposition (4 + k, 1);
    _outtext ("Enter 'S' to save new A-to-D scale factor to non-volatile");

    _settextposition (5 + k, 1);
    _outtext ("memory on AD1170's : ");

    gets (string);

    if (string[0] == 'S' || string[0] == 's')
      {
        wait_for_A ();
        outp (COMMAND_REG_A, SAVA);

        wait_for_B ();
        outp (COMMAND_REG_B, SAVA);
      }
  }


void calibrate_QA2000 (void)
  {
    char
        test_str[80];

    double
     start_p,  dist,
     end_p,      calc_p;


    _setbkcolor ((long) 1);
    _settextcolor ((short) 15);
    _clearscreen (_GCLEARSCREEN);

    command_anorad ("A5.");

    wait_for (6.0);

    start_p = read_position ();

    bias = bias + do_zupt (ZUPT_POINTS);
    SF = 1.0;
                              calc_p       =       integrate_and_move       (4.0,
"D00D001881D01FF00000000D0FE77E30000000000G");

    end_p = read_position ();
    dist = end_p - start_p;

    SF = dist/calc_p;
    bias_time = 0.0;
    printf ("\ndist = %g  SF = %g     bias = %g", dist, SF, bias);
    gets (test_str);
  }


void command_anorad (char *ano_buffer)
  {
    int
        i;

    for (i = 0; ano_buffer[i] != '\0'; ++i)
      {
        _bios_serialcom (_COM_SEND, 0, (unsigned int) ano_buffer[i]);
      }
  }



/* Change the cursor shape.
   <value> has starting line in upper byte, ending line in lower byte.
   Returns the previous  cursor value. */

unsigned cursor (unsigned value)
  {
    union REGS inregs, outregs;
    int ret;

    inregs.h.ah = 3;        /* Get old cursor */
    inregs.h.bh = 0;
    int86 (0x10,&inregs,&outregs);
    ret = outregs.x.cx;

    inregs.h.ah = 1;        /* Set new cursor */
    inregs.x.cx = value;
    int86 (0x10,&inregs,&outregs);

    return (ret);
  }



/* Put an item in menu.
```

```
   <row> and <col> are left position.
   <str> is the string item.
   <len> is the number of blanks to fill. */

void display_menu_item (int left_row, int left_col, char *str, int len)
  {
    char
        temp[80];

    _settextposition (left_row, left_col);
    _outtext (" ");
    _outtext (str);
    memset (temp,' ',len--);
    temp[len] = 0;
    _outtext (temp);
  }



double do_zupt (int numb_data_points)
  {
    int
        i;

    double
        accel_sum;

    accel_sum = 0.0;

    for (i = 0; i < numb_data_points/2; ++i)
      {
        accel_sum = accel_sum + read_A () - bias;
        accel_sum = accel_sum + read_B () - bias;
      }
    return (accel_sum/(double) numb_data_points);
  }



/* Draw   menu box.
   <row> and <col> are upper left of box.
   <hi> and <wid> are height and width. */

void draw_menu_box (int up_left_row, int up_left_col, int height, int width)
  {
    int
        i;
    char
        temp[80];

    _settextposition (up_left_row, up_left_col);
    temp[0] = *menus.nw;
    memset (temp + 1, *menus.ew, width);
    temp[width + 1] = *menus.ne;
    temp[width + 2] = 0;
    _outtext (temp);

    for (i = 1; i <= height; ++i)
      {
        _settextposition (up_left_row + i, up_left_col);
        _outtext (menus.ns);
        _settextposition (up_left_row + i, up_left_col + width + 1);
        _outtext (menus.ns);
      }

    _settextposition (up_left_row + height + 1, up_left_col);
    temp[0] = *menus.sw;
    memset (temp + 1, *menus.ew, width);
    temp[width + 1] = *menus.se;
    temp[width + 2] = 0;
    _outtext (temp);
  }



double exper_1 (void)
  {
    double
     start_p,  end_p,  calc_p,  err,  data_time;


    data_time = 3.0;

    command_anorad ("A5.");

    wait_for (8.0);

    outp (CNTR_STOP, 0x0000);       /* Disarm 2MCLK  */

    wait_for_A ();
    outp (COMMAND_REG_A, SCAL);

    wait_for_B ();
    outp (COMMAND_REG_B, SCAL);

    wait_for_A ();
    wait_for_B ();

    outp (CNTR_START, 0x0000);            /* Arm 2MCLK */

    start_p = read_position ();

                        calc_p      =      integrate_and_move      (data_time,
"D00D001881D01FF00000000D0FE77E30000000000G");

    end_p = read_position ();

    err = calc_p - (end_p - start_p);

    fprintf (log_ptr, "\n%14.6g    %14.6g   %14.6g",
          calc_p, end_p - start_p, err);

    return (err);
  }



double exper_2 (void)
  {
    double
     start_p,  end_p,  calc_p,  err,  data_time;
    data_time = 3.0;

    command_anorad ("A5.");

    wait_for (8.0);

    outp (CNTR_STOP, 0x0000);       /* Disarm 2MCLK  */

    wait_for_A ();
    outp (COMMAND_REG_A, SCAL);

    wait_for_B ();
    outp (COMMAND_REG_B, SCAL);

    wait_for_A ();
    wait_for_B ();

    outp (CNTR_START, 0x0000);            /* Arm 2MCLK */

    start_p = read_position ();

    bias = bias + do_zupt (ZUPT_POINTS);

                        calc_p     =     integrate_and_move      (data_time,
"D00D001881D01FF00000000D0FE77E30000000000G");

    wait_for (2.0);

    bias = bias + do_zupt (ZUPT_POINTS);

    calc_p = calc_p +
                                          integrate_and_move       (data_time,
"D00D001881D01FF00000000D0FE77E30000000000G");

    wait_for (2.0);

    bias = bias + do_zupt (ZUPT_POINTS);

    calc_p = calc_p +
                                          integrate_and_move       (data_time,
"D00D001881D01FF00000000D0FE77E30000000000G");

    end_p = read_position ();

    err = calc_p - (end_p - start_p);

    fprintf (log_ptr, "\n%14.6g    %14.6g   %14.6g",
          calc_p, end_p - start_p, err);

    return (err);
  }



double exper_3 (void)
  {
    double
     start_p,  end_p,  calc_p,  err,  data_time,  vel_sum;

    data_time = 3.0;

    command_anorad ("A5.");
    wait_for (8.0);

    outp (CNTR_STOP, 0x0000);        /* Disarm 2MCLK  */

    wait_for_A ();
    outp (COMMAND_REG_A, SCAL);

    wait_for_B ();
    outp (COMMAND_REG_B, SCAL);

    wait_for_A ();
    wait_for_B ();

    outp (CNTR_START, 0x0000);             /* Arm 2MCLK */

    start_p = read_position ();

                         calc_p      =      integrate_and_move      (data_time,
"D00D001881D01FF00000000D0FE77E30000000000G");
    vel_sum = vel;

    wait_for (2.0);

    calc_p = calc_p +
          integrate_and_move (data_time, "A5.");
    vel_sum = vel_sum + vel;

    end_p = read_position ();

    err = calc_p - (end_p - start_p);

                     bias      =      bias      -      (2.0*vel_sum/data_time      +
6.0*err/(data_time*data_time))/SF;
    bias_time = bias_time + (6.0*vel_sum/(data_time*data_time) -
            12.0*err/(data_time*data_time*data_time))/SF;

    fprintf (log_ptr, "\n%14.6g    %14.6g   %14.6g",
          calc_p, end_p - start_p, err);

    return (err);
  }



double exper_4 (void)
  {
    double
     start_p,  end_p,  calc_p,  err,  data_time,  vel_sum;

    data_time = 3.0;

    command_anorad ("A5.");

    wait_for (8.0);

    outp (CNTR_STOP, 0x0000);        /* Disarm 2MCLK  */

    wait_for_A ();
    outp (COMMAND_REG_A, SCAL);

    wait_for_B ();
    outp (COMMAND_REG_B, SCAL);

    wait_for_A ();
    wait_for_B ();

    outp (CNTR_START, 0x0000);           /* Arm 2MCLK */

    start_p = read_position ();

                         calc_p      =      integrate_and_move      (data_time,
"D00D001881D01FF00000000D0FE77E30000000000G");
    vel_sum = vel;

    wait_for (2.0);

    calc_p = calc_p +
          integrate_and_move (data_time, "A5.");
    vel_sum = vel_sum + vel;

    end_p = read_position ();

    err = calc_p - (end_p - start_p);

                     bias     =     bias     -     (2.0*vel_sum/data_time     +
6.0*err/(data_time*data_time))/SF;
          bias_time   =   bias_time   +   (6.0*vel_sum/(data_time*data_time)   -
12.0*err/(data_time*data_time*data_time))/SF;

    fprintf (log_ptr, "\n%14.6g    %14.6g   %14.6g",
          calc_p, end_p - start_p, err);

    return (err);
  }



double exper_5 (void)
  {
    double
     start_p,  end_p,  calc_p,  err,  data_time,  vel_sum;

    data_time = 3.0;

    command_anorad ("A5.");

    wait_for (8.0);

    outp (CNTR_STOP, 0x0000);       /* Disarm 2MCLK  */

    wait_for_A ();
    outp (COMMAND_REG_A, SCAL);

    wait_for_B ();
    outp (COMMAND_REG_B, SCAL);

    wait_for_A ();
    wait_for_B ();

    outp (CNTR_START, 0x0000);           /* Arm 2MCLK */

    start_p = read_position ();
```

```
                             calc_p       =       integrate_and_move       (data_time,
"D00D001881D01FF00000000D0FE77E30000000000G");
    vel_sum = vel;

    wait_for (2.0);

    calc_p = calc_p +
                                                  integrate_and_move       (data_time,
"D00D001881D01FF00000000D0FE77E30000000000G");
    vel_sum = vel_sum + vel;

    end_p = read_position ();

    err = calc_p - (end_p - start_p);

                         bias      =      bias      -      (2.0*vel_sum/data_time      +
6.0*err/(data_time*data_time))/SF;
          bias_time    =   bias_time   +   (6.0*vel_sum/(data_time*data_time)   -
12.0*err/(data_time*data_time*data_time))/SF;

    SF = SF*end_p/calc_p;

    fprintf (log_ptr, "\n%14.6g    %14.6g   %14.6g",
          calc_p, end_p - start_p, err);

    return (err);
  }



double exper_6 (void)
  {
    double
     start_p,  end_p,  calc_p,  err,  data_time;

    data_time = 3.0;

    command_anorad ("A5.");

    wait_for (8.0);

    outp (CNTR_STOP, 0x0000);       /* Disarm 2MCLK  */

    wait_for_A ();
    outp (COMMAND_REG_A, SCAL);

    wait_for_B ();
    outp (COMMAND_REG_B, SCAL);

    wait_for_A ();
    wait_for_B ();

    outp (CNTR_START, 0x0000);            /* Arm 2MCLK */

    start_p = read_position ();

    bias = bias + do_zupt (ZUPT_POINTS);

                             calc_p       =       integrate_and_move       (data_time,
"D00D001881D01FF00000000D0FE77E30000000000G");

    wait_for (2.0);
    bias = bias + do_zupt (ZUPT_POINTS);

    calc_p = calc_p +
                                        integrate_and_move        (data_time,
"D00D001881D01FF00000000D0FE77E30000000000G");

    end_p = read_position ();

    err = calc_p - (end_p - start_p);

    SF = (SF + SF*end_p/calc_p)/2.0;

    fprintf (log_ptr, "\n%14.6g    %14.6g   %14.6g",
          calc_p, end_p - start_p, err);

    return (err);
  }



void free_run_mode (void)
  {
    char
        string [80];

    int
        in_char;

    unsigned int
        ret;

    double
     accel_m1,  accel_m2,  accel_m3,  accel,  position,  d_cnt,
     vel_m1,    vel_m2,    vel_m3,  tot_time,  p_correct,
     new_bias,  slope;


    _setbkcolor ((long) 1);
    _settextcolor ((short) 15);
    _clearscreen (_GCLEARSCREEN);

    accel_m1 = 0.0;
    accel_m2 = 0.0;
    accel_m3 = 0.0;
    accel = 0.0;

    vel_m1 = 0.0;
    vel_m2 = 0.0;
    vel_m3 = 0.0;
    vel = 0.0;

    d_cnt = 0.0;

    position = read_position ();

    do
      {
        if (kbhit() != 0)
          {
            in_char = getch ();
            if (in_char == (int) '*')
```

```
         {
           return;
      }
    if (in_char == (int) 'Z')
      {
     new_bias = bias + do_zupt (ZUPT_POINTS);
     tot_time = d_cnt*dt*2.0;
     slope = SF*(new_bias - bias)/tot_time;
     p_correct = slope*tot_time*tot_time*tot_time/6.0;
     position = position-p_correct;

     bias = new_bias;

     d_cnt = 0.0;

     accel_m1 = 0.0;
     accel_m2 = 0.0;
     accel_m3 = 0.0;
     accel = 0.0;

     vel_m1 = 0.0;
     vel_m2 = 0.0;
     vel_m3 = 0.0;
     vel = 0.0;
      }
    else
      {
     ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) in_char);
      }
  }

   accel_m3 = accel_m2;
   accel_m2 = accel_m1;
   accel_m1 = accel;
accel = SF*(read_A () - bias);

   vel_m3 = vel_m2;
   vel_m2 = vel_m1;
   vel_m1 = vel;

   switch (integ_alg)
     {
       case SIMP:
         vel = vel + (5.0*accel + 8.0*accel_m1 - accel_m2)*dt_12;
         position = position + (5.0*vel + 8.0*vel_m1 - vel_m2)*dt_12;
       break;

       case TRAP:
         vel = vel + (accel + accel_m1)*dt_2;
         position = position + (vel + vel_m1)*dt_2;
       break;

       case ADAM:
         vel = vel + (55.0*accel - 59.0*accel_m1 +
                      37.0*accel_m2 - 9.0*accel_m3)*dt_24;
         position = position + (55.0*vel - 59.0*vel_m1 +
                      37.0*vel_m2 - 9.0*vel_m3)*dt_24;
       break;
     }

   accel_m3 = accel_m2;
   accel_m2 = accel_m1;
   accel_m1 = accel;
```

```
     accel = SF*(read_B () - bias);

        vel_m3 = vel_m2;
        vel_m2 = vel_m1;
        vel_m1 = vel;

        switch (integ_alg)
          {
            case SIMP:
              vel = vel + (5.0*accel + 8.0*accel_m1 - accel_m2)*dt_12;
              position = position + (5.0*vel + 8.0*vel_m1 - vel_m2)*dt_12;
            break;

            case TRAP:
              vel = vel + (accel + accel_m1)*dt_2;
              position = position + (vel + vel_m1)*dt_2;
            break;

            case ADAM:
              vel = vel + (55.0*accel - 59.0*accel_m1 +
                           37.0*accel_m2 - 9.0*accel_m3)*dt_24;
              position = position + (55.0*vel - 59.0*vel_m1 +
                           37.0*vel_m2 - 9.0*vel_m3)*dt_24;
            break;
          }

     sprintf (string, "Position = %12.3f inches", position);
     _settextposition (center_row, center_col - 16);
     _outtext (string);

     d_cnt = d_cnt + 1.0;
      }
    while (1);
  }



double integrate_and_move (double seconds, char *ano_buffer)
  {
    int
        string_len,  i,  k,  numb_points;

    double
        accel_m1,  accel_m2,  accel_m3,  accel,  position,
        vel_m1,    vel_m2,    vel_m3;


    _setbkcolor ((long) 1);
    _settextcolor ((short) 15);
    _clearscreen (_GCLEARSCREEN);

    string_len = strlen (ano_buffer);
    k = 0;
    numb_points = (int) (seconds/dt);

    accel_m1 = 0.0;
    accel_m2 = 0.0;
    accel_m3 = 0.0;
    accel = 0.0;

    vel_m1 = 0.0;
    vel_m2 = 0.0;
```

```
vel_m3 = 0.0;
vel = 0.0;

position = 0.0;

for (i = 0; i < numb_points/2; ++i)
  {
    accel_m3 = accel_m2;
    accel_m2 = accel_m1;
    accel_m1 = accel;
    accel = SF*(read_A () - bias - bias_time*dt*i);

    vel_m3 = vel_m2;
    vel_m2 = vel_m1;
    vel_m1 = vel;

    switch (integ_alg)
      {
        case SIMP:
          vel = vel + (5.0*accel + 8.0*accel_m1 - accel_m2)*dt_12;
          position = position + (5.0*vel + 8.0*vel_m1 - vel_m2)*dt_12;
        break;

        case TRAP:
          vel = vel + (accel + accel_m1)*dt_2;
          position = position + (vel + vel_m1)*dt_2;
        break;

        case ADAM:
          vel = vel + (55.0*accel - 59.0*accel_m1 +
                       37.0*accel_m2 - 9.0*accel_m3)*dt_24;
          position = position + (55.0*vel - 59.0*vel_m1 +
                       37.0*vel_m2 - 9.0*vel_m3)*dt_24;
        break;
      }

    accel_m3 = accel_m2;
    accel_m2 = accel_m1;
    accel_m1 = accel;
 accel = SF*(read_B () - bias - bias_time*dt*i);

    vel_m3 = vel_m2;
    vel_m2 = vel_m1;
    vel_m1 = vel;

    switch (integ_alg)
      {
        case SIMP:
          vel = vel + (5.0*accel + 8.0*accel_m1 - accel_m2)*dt_12;
          position = position + (5.0*vel + 8.0*vel_m1 - vel_m2)*dt_12;
        break;

        case TRAP:
          vel = vel + (accel + accel_m1)*dt_2;
          position = position + (vel + vel_m1)*dt_2;
        break;

        case ADAM:
          vel = vel + (55.0*accel - 59.0*accel_m1 +
                       37.0*accel_m2 - 9.0*accel_m3)*dt_24;
          position = position + (55.0*vel - 59.0*vel_m1 +
                       37.0*vel_m2 - 9.0*vel_m3)*dt_24;
        break;
```

```
          }

        if (k < string_len)
          {
            _bios_serialcom (_COM_SEND, 0, (unsigned int) ano_buffer[k]);
         ++k;
          }
      }
    return (position);
  }



/* Put menu on screen.
   Starting <row> and <column>.
   Array of menu <items> strings.
   Global structure variable <menus> determines:
   Colors of border, normal items, and selected item.
   Centered or left justfied.
   Border characters.
   Returns number of item selected. */

int get_menu (int row, int col, char * *items)
  {
    int
        i, numb_items, max = 2, prev, curr = 0,
        litem[25];

    long
        bcolor;

    cursor (TCURSOROFF);
    bcolor = _getbkcolor ();

    /* Count items, find longest, and put length of each in array */

    for (numb_items = 0; items[numb_items] != NULL; numb_items++)
      {
        litem[numb_items] = strlen (items[numb_items]);
        if (max < litem[numb_items])
          max = litem[numb_items];
      }
    max = max + 2;

    if (menus.centered)
      {
        row -= numb_items / 2;
        col -= max / 2;
      }

    /* Draw menu box */

    _settextcolor (menus.fgBorder);
    _setbkcolor (menus.bgBorder);
    draw_menu_box (row++,col++,numb_items,max);

    /* Put items in menu */

    for (i = 0; i < numb_items; ++i)
      {
        if (i == curr)
          {
            _settextcolor (menus.fgSelect);
            _setbkcolor (menus.bgSelect);
          }
        else
          {
            _settextcolor (menus.fgNormal);
            _setbkcolor (menus.bgNormal);
          }
        display_menu_item (row+i,col,items[i],max - litem[i]);
      }

    /* Get selection */

    for (;;)
      {
        switch ((_bios_keybrd(_KEYBRD_READ) & 0xff00) >> 8)
          {
            case UP :
              prev = curr;
              curr = (curr > 0) ? (--curr % numb_items) : numb_items-1;
            break;

            case DOWN :
              prev = curr;
              curr = (curr < numb_items) ? (++curr % numb_items) : 0;
            break;

            case ENTER :
              _setbkcolor (bcolor);
              return (curr);

            default :
              continue;
          }
        _settextcolor (menus.fgSelect);
        _setbkcolor (menus.bgSelect);
        display_menu_item (row + curr, col, items[curr], max - litem[curr]);

        _settextcolor (menus.fgNormal);
        _setbkcolor (menus.bgNormal);
        display_menu_item (row + prev, col, items[prev], max - litem[prev]);
      }
  }



double read_A ()
  {
    int
        high_byte,  mid_byte,  low_byte,  status_byte;

    long int
        h_byte,     m_byte,    l_byte,    adc_result;

    double
        adc_double;
    do
      {
        status_byte = inp (STATUS_REG_A);
      }
    while ((status_byte & 0x0002) == 0x0000);
```

```
   /***************************************************************/
   /*  Read low, mid and high bytes                               */
   /***************************************************************/

    high_byte = inp (HIGH_DATA_A);
    mid_byte = inp (MID_DATA_A);
    low_byte = inp (LOW_DATA_A);

   /***************************************************************/
   /*  Combine bytes                                              */
   /***************************************************************/

    h_byte = (long int) high_byte;
    m_byte = (long int) mid_byte;
    l_byte = (long int) low_byte;
    adc_result = ((h_byte << 16) & 0xffff0000) |
                 ((m_byte << 8) & 0x0000ff00) |
                  (l_byte & 0x000000ff);

    adc_double = (10.0*adc_result)/4194304.0 - 5.0;
    return (adc_double);
  }



double read_B ()
  {
    int
        high_byte,  mid_byte,  low_byte,  status_byte;

    long int
        h_byte,     m_byte,    l_byte,   adc_result;

    double
        adc_double;

    do
      {
        status_byte = inp (STATUS_REG_B);
      }
    while ((status_byte & 0x0002) == 0x0000);

   /***************************************************************/
   /*  Read low, mid and high bytes                               */
   /***************************************************************/

    high_byte = inp (HIGH_DATA_B);
    mid_byte = inp (MID_DATA_B);
    low_byte = inp (LOW_DATA_B);

   /***************************************************************/
   /*  Combine and store in array                                 */
   /***************************************************************/

    h_byte = (long int) high_byte;
    m_byte = (long int) mid_byte;
    l_byte = (long int) low_byte;
    adc_result = ((h_byte << 16) & 0xffff0000) |
                 ((m_byte << 8) & 0x0000ff00) |
                  (l_byte & 0x000000ff);

    adc_double = (10.0*adc_result)/4194304.0 - 5.0;
    return (adc_double);
  }



double read_position ()
 {
  unsigned int
      com1_ret,   com1_ret1,   com1_ret2,   com1_ret3,            com1_ret4,
com1_ret5,
      com1_ret6,  com1_ret7;

  double
      posi;

  do
    {
      com1_ret7 = _bios_serialcom (_COM_RECEIVE, 0, 0);
      com1_ret7 = _bios_serialcom (_COM_RECEIVE, 0, 0);
      com1_ret7 = _bios_serialcom (_COM_RECEIVE, 0, 0);

      com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) 'Q');

      com1_ret1 = _bios_serialcom (_COM_RECEIVE, 0, 0);
      com1_ret2 = _bios_serialcom (_COM_RECEIVE, 0, 0);
      com1_ret3 = _bios_serialcom (_COM_RECEIVE, 0, 0);
      com1_ret4 = _bios_serialcom (_COM_RECEIVE, 0, 0);
      com1_ret5 = _bios_serialcom (_COM_RECEIVE, 0, 0);
      com1_ret6 = _bios_serialcom (_COM_RECEIVE, 0, 0);
      com1_ret7 = _bios_serialcom (_COM_RECEIVE, 0, 0);

      if (com1_ret1 < 65)
     {
       posi = 1048576.0 * ((double) com1_ret1 - 48);
     }
      else
     {
       posi = 1048576.0 * ((double) com1_ret1 - 55);
     }

      if (com1_ret2 < 65)
     {
       posi = posi + 65536.0 * ((double) com1_ret2 - 48);
     }
      else
     {
       posi = posi + 65536.0 * ((double) com1_ret2 - 55);
     }

      if (com1_ret3 < 65)
     {
       posi = posi + 4096.0 * ((double) com1_ret3 - 48);
     }
      else
     {
       posi = posi + 4096.0 * ((double) com1_ret3 - 55);
     }

      if (com1_ret4 < 65)
     {
       posi = posi + 256.0 * ((double) com1_ret4 - 48);
     }
```

```
      else
     {
       posi = posi + 256.0 * ((double) com1_ret4 - 55);
     }

      if (com1_ret5 < 65)
     {
       posi = posi + 16.0 * ((double) com1_ret5 - 48);
     }
      else
     {
       posi = posi + 16.0 * ((double) com1_ret5 - 55);
     }

      if (com1_ret6 < 65)
     {
       posi = posi + 1.0 * ((double) com1_ret6 - 48);
     }
      else
     {
       posi = posi + 1.0 * ((double) com1_ret6 - 55);
     }

      posi = posi/64000.0;
    }
  while (posi > 12.0 || posi < 0.0);

  return (posi);
 }



void setup_ADC_defaults (void)
  {
    wait_for_A ();
    outp (COMMAND_REG_A, RST);

    wait_for_A ();
    outp (COMMAND_REG_A, SDC | INTEG_300_M);

    wait_for_B ();
    outp (COMMAND_REG_B, RST);

    wait_for_B ();
    outp (COMMAND_REG_B, SDC | INTEG_300_M);

    wait_for_A ();
    outp (PARAM_1_REG_A, 0x000f);
    outp (COMMAND_REG_A, SDF);

    wait_for_B ();
    outp (PARAM_1_REG_B, 0x000f);
    outp (COMMAND_REG_B, SDF);

    wait_for_A ();
    outp (COMMAND_REG_A, CALDI);

    wait_for_A ();
    outp (COMMAND_REG_A, NULDI);

    wait_for_A ();
    outp (COMMAND_REG_A, SCAL);
```

```
void setup_anorad (void)
 {
  unsigned int
      com1_ret;

  com1_ret = _bios_serialcom (_COM_INIT, 0,
            _COM_9600 | _COM_EVENPARITY | _COM_CHR7 | _COM_STOP2);

  com1_ret = _bios_serialcom (_COM_SEND, 0, 1);

  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) 'M');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) 'H');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) '2');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) '5');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) '6');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) '.');

  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) 'H');

  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) 'E');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) '1');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) '.');

  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) 'F');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) '2');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) '0');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) '0');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) '.');

  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) 'M');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) 'A');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) '5');
  com1_ret = _bios_serialcom (_COM_SEND, 0, (unsigned int) '.');

  com1_ret = _bios_serialcom (_COM_RECEIVE, 0, 0);
  com1_ret = _bios_serialcom (_COM_RECEIVE, 0, 0);
  com1_ret = _bios_serialcom (_COM_RECEIVE, 0, 0);
 }



double setup_counter (double frequency)
  {
    unsigned int
     numb_counts,   chigh,      clow;

    double
     actual_freq;

    /*************************************************************/
    /*    Program counters on ADC board                 */
    /*************************************************************/

    numb_counts = (unsigned int)(2.0E+06/frequency);
    chigh = (numb_counts >> 8) & 0x00ff;
    clow = numb_counts & 0x00ff;

    outp (CNTR_STOP, 0x0000);
    outp (CNTR_LOW_BYTE, clow);
    outp (CNTR_HIGH_BYTE, chigh);
    actual_freq = 2.0E+06/((double) numb_counts);
    return (actual_freq);
  }



void wait_for (double seconds)
  {
    char
        string[80];

    int
        i,  numb_points;

    double
        input;


    _setbkcolor ((long) 1);
    _settextcolor ((short) 15);
    _clearscreen (_GCLEARSCREEN);

    _settextposition (center_row, center_col - 15);
    sprintf (string, "Waiting for %10.2g seconds", seconds);
    _outtext (string);

    numb_points = (int) (seconds/dt);

    for (i = 0; i < numb_points/2; ++i)
      {
        input = read_A ();
        input = read_B ();
      }
  }



void wait_for_A ()
 {
  unsigned int
    status_word;

  do
   {
    status_word = inp (STATUS_REG_A);
   }
  while ((status_word & 0x0001) == 1 || (status_word & 0x0020) == 0x0020);

 }



void wait_for_B ()
 {
  unsigned int
    status_word;

  do
   {
```

```
   status_word = inp (STATUS_REG_B);
  }
 while ((status_word & 0x0001) == 1 || (status_word & 0x0020) == 0x0020);

}
```

Appendix D - Anorad Table Commands

IAC PROGPAMMING CODES EXPLANATION

This section contains detailed descriptions of all commands. These fall into two groups: Buffered (non-immediate) and Non-Buffered (immediate).

Buffered Commands

The following commands are stored by the axis; they are executed in sequence immediately following the completion of the previous command (see Enable Synchronized Ready command page 2-12 for more information). All require that the axis be addressed in order to be accepted.

Data Input Commands

The format of all data input commands (except "D") is:

```
    I  [n..n]  t
    :     :    :
    :     :     -terminator -  any non digit character excluding
    :     :                    address characters.  Can be the next
    :     :                    command letter.  In the following
    :     :                    examples "." is used.
    :     :
    :      ------------data -  from one to seven digits.  The []
    :                          indicate that this data is optional;
    :                          if omitted then a command value of
    :                          zero is used.
    :
     --------command-letter -  a single letter.
```

A - Absolute Position Command

Commands the axis in Point-to-Point mode to a specific position relative to zero.

Ex:	A10000.	move to +10000 counts
	A-1234.	move to -1234 counts

D - Acceleration, Deceleration Data

Used for entering data to define a controlled-path move. This is done by commanding the axes to accelerate at given rate for a given time. Acceleration is in counts per interrupt units, and time is in .498 millisecond units. Each frame consists of ten hexadecimal digits: ttttiiffff where tttt is the frame time, ii is the integer portion of the acceleration (counts) and ffff is the fractional portion of the acceleration (1/65536 counts). The data input is terminated (at the end of a frame) by either a non-hexadecimal character or a frame with zero time. A frame with zero time indicates the end of the contour, at which the axis switches to Point-to-Point mode. (See interpolation description for format and a detailed example.)

```
                    Ex:  D00080080000008FF80000000000000
                          ttttiiffffttttiiffffttttiiffff
    actually:  D 0008 00 8000   accel at .5cts/int for 8 interrupts,
                 0008 FF 8000   decel at -.5 cts/int for 8 int,
                 0000 00 0000   zero frame time - end of contour
```

E - Set Early Ready Distance

Sets the position tolerance in counts within which the axis is said to have reached its commanded position. That is, when the axis is within "E" counts of its commanded position, its "ready" status will become true. The default (power-on) early ready distance is 13 counts (approx .0002" for EE1 encoder); the maximum allowable value is 255 counts.

```
    Ex:  E2.       set early ready distance to 2 counts
         E100.     set early ready distance to 100 counts
         E0.       set no early ready distance - axis must reach
                   commanded position to be ready
```

F - Set Maximum Velocity

Used to specify the maximum velocity in Point-to-Point mode. The data is in units of .1% of maximum velocity, that is, F1000 is maximum velocity, F100 is one-tenth of maximum, etc.

```
    Ex:  F1000.        set maximum point-to-point vel.
         F10.          set 1% of maximum vel.
```

I - Incremental Position Command

Commands the axis to move a specified distance (increment) from its current position. This is in contrast to absolute, which commands the axis to a specific position, instead of by a specific distance.

```
    Ex:  I1200.        move the axis +1200 counts
         I-123.        move the axis -123 counts
```

O - Home Offset

This command is used to implement coordinate translation, i.e. re-defining the zero reference position.

```
    Ex:  O64000.       set axis zero to +64000 counts from home (the
                       axis' position is now -64000)
         O0.           set axis zero to home (clear the offset)
         O-20000.      set axis zero to -20000 counts from home (the
                       axis' position is now +20000)
```

Other Commands

H - Home

Causes the axis to begin the Home/Calibrate sequence in which the encoder signals are first sampled and then compensated for, after which the axis' machine home position (according to mechanical or optical sensors) is determined. (See standard encoder home sequence.)

Ex. H Homes axis

Z - Zero Set (reset)

Introduces an offset equal to the value of the current absolute position, thereby defining that position as zero. (Independent of Home Offset "O".)

Ex. Z Zeros axis
Q command will display 000000.

M - Select Variable Axis Parameters

Allows the user to configure the axis for particular tables, loads, etc. The command requires an extra letter to select the parameter to be changed.

--MA - Select Square Root Deceleration

Used to select square root (constant) deceleration in pt-to-pt positioning mode. The command requires a single digit from 0 to 11 to select deceleration according to the following table. (Note - the given deceleration values are based upon a standard EE1 encoder i.e. 15.625u"/count = 1/64000"/ct, and maximum table velocity of 10 in/sec)

MA0	=	1/512g	=	.7539"/sec/sec
MA1	=	1/256g	=	1.508"/sec/sec
MA2	=	1/128g	=	3.016"/sec/sec
MA3	=	1/64g	=	6.02"/sec/sec
MA4	=	1/32 g	=	12.063"/sec/sec
MA5	=	1/16 g	=	24.125"/sec/sec
MA6	=	1/8 g	=	48.25"/sec/sec
MA7	=	1/4 g	=	96.5"/sec/sec
MA8	=	1/2 g	=	193"/sec/sec
MA9	=	1 g	=	386"/sec/sec
MA10	=	2 g	=	772"/sec/sec
MA11	=	4 g	=	1544"/sec/sec

Ex: MA7. select 1/4 g deceleration
MA4. select 1/32 g deceleration

MC - MA for lowspeed mode. Tach switching systems only, see addendum

MD - SET HOME SEARCH DISTANCE

Sets half the number of cycles searched for the home pulse. (The number of cycles searched is twice what is entered by MD).

Ex:	MD25.	For 500 line per inch encoder
	MD31.	For 25 line per mm encoder
	MD125.	For 2500 line per inch or 100 line per mm encoder

MF - DAC Bias Compensation

This number is added to the DAC output when in the linear region of the point to point servo. The purpose of this is to insure that the output is enough to overcome a small DAC offset.

EX.:	MF 4.	set to 4 DAC counts
	MF 0	set to 0 DAC counts
	MF 10	set to 10 DAC counts

MG - Interpolation Error Gain

MG0 = Positional Error / 128

MG1 = Positional Error / 64

MG2 = Positional Error / 32

MG3 = Positional Error / 16

MG4 = Positional Error / 8

MG5 = Positional Error / 4

MG6 = Positional Error / 2

MG7 = Positional Error / 1

--MH - Set Home Speed

Used to set the maximum speed at which the axis will search for the home reference. The speed is programmed in D/A counts, where maximum is 4095.

Ex: MH4095. set maximum home speed

MH1024. set 1/4 maximum speed

MI - Desired Velocity (Interpolation) Scale Factor

MI0	= x	1
MI1	= x	2
MI2	= x	3
MI3	= x	4
MI4	= x	6
MI5	= x	8
MI6	= x	12
MI7	= x	16
MI8	= x	24
MI9	= x	32
MI10	= x	48
MI11	= x	64
MI12	= x	96
MI13	= x	128

DVEL scale factor = 4095 (max DAC) / Maximum System Speed In cts/Int

--MR - Select Linear Deceleration Slope

Used to select the linear (declining) deceleration slope. It is typically used to match performance and stability for a given axis. (See Point-to-Point servo description)

```
MR0 --    vel = 1/16 x dist to goal
MR1 --    vel = 1/8  x dist to goal
MR2 --    vel = 1/4  x dist to goal
MR3 --    vel = 1/2  x dist to goal
MR4 --    vel =   1  x dist to goal
MR5 --    vel =    2 x dist to goal
MR6 --    vel =    4 x dist to goal

Ex:  MR0.    select 1/16 linear slope
     MR3.    select 1/2 linear slope
```

Non-Buffered (Immediate) Commands

The following commands are acted upon as soon as they are received by the axis.

Addressed Commands (require axis to be addressed)

Q - Output Position

Causes the current position of the axis to be output over the communications interface. The position is transmitted as six hexadecimal characters (3 bytes) and is two's-compliment binary.

```
             :              :
char1  char2 : char3  char4 : char5  char6 & CR
  :      :   :   :      :   :   :      :
  :      :       :      :        ------------least significant byte
  :      :       :      :
  :      :        ---------------------------middle byte
  :      :
   ------------------------------------------most significant byte
```

Ex. Q Output position
00000F response - 16 counts positive from 0 position

U - Enable Synchronized Ready Mode

Causes the axis to link itself to the Universal Ready signal. This signal is the logical "AND" of all linked axes' ready states; it indicates that all linked axes have completed command execution. When linked in this manner, the axes begin each new command at the same time, allowing for synchronized system control.

Ex. U Enables synchronized ready mode.

V - Disable Synchronized Ready Mode

Disconnects the axis from the Universal Ready Signal.

Ex. V Disables synchronized ready mode.

COMMAND SUMMARY

			default	data	immed	addr
A	Absolute Position Command	Asnnnnnnn		..x..		..x
D	Interpolation Acceleration Data	Dhhhhhhhhhh		..x..		..x
E	Set Early Ready Distance	Ennn	13	..x..		..x
F	Set Maximum Velocity	Fnnnn	250	..x..		..x
G	Acceleration Servo Mode	G		.. ..		..x
H	Home	H		.. ..		..x
I	Incremental Position Command	Isnnnnnnn		..x..		..x
J	Servo Off	J	J	.. ..		..x
M	Select variable axis parameters:					
MA	Select pt-pt deceleration	MAn	6	..x..		..x
MB	Set Velocity Offset	MBsnnn	0	..x..		..x
MC	MA for low speed mode Tach switching systems only.	MCn	0	..x..		..x
MD	Set Home Pulse Search Dist.	MDnn	+50 cycles	..x..		..x
MF	Set DAC Bias Compensation	MFn	2	..x..		..x
MG	Interpolation Error Gain	MGn	4	..x..		..x
MH	Set Home Speed	MHnnnn	1,000	..x..		..x
MI	Interp.Des. Vel. Scale Fac.	MIn	6	..x..		..x
ML	Set Lin (usually 2X Predis)	MLnnnnn		.. ..		..
MP	Set Predis	MPnnnnn		.. ..		..
MR	Select Linear Slope	MRn	0	..x..		..x
O	Set Home Offset	Osnnnnnnn	0	..x..		..x
P	Point-to-Point Servo Mode	P		.. ..		..x
Q	Output Position	Q		.. ..	x	..x
R	Servo Off	R		.. ..		..x
U	Enable Universal Ready Mode	U		.. ..	x	..x
V	Disable Universal Ready Mode	V	V	.. ..	x	..x
W	Output Status	W		.. ..	x	..x
X	Abort Command Execution	X		.. ..	x	..
Y	Abort Command Execution	Y		.. ..	x	..x
Z	Set Zero	Z		.. ..		..x

"data" indicates that command requires numeric data

"immed" indicates that command is executed as soon as it's received

"addr" indicates that axis must be addressed for command to be executed

n = decimal digit 0-9

h = hexadecimal digit 0-9, A-F

s = sign (+,-); if omitted, + assumed

NOTE: For the data input commands, the number of digits shown is the maximum number of digits allowed - any smaller number of digits is acceptable; only significant digits are necessary.

```
                          ZUPT W/2 BASES (M2)

          7.30441              7.15834          0.146061
           7.1865              7.15834         0.0281605
          7.18247              7.15834         0.0241231
          7.15927              7.15833       0.000940704

   Mean Error =       0.04982 (inches)    STD =       0.05653 (inches)
   Max Error =        0.1461 (inches)
   Sampling rate = 300.03 (Hz)
   Filter bandwidth = 40 (Hz)
   A-to-D Range = 1 (Gs)
   Integration method = 1
         (0-Simp  1-Trap  2-Adam)
   Number of runs = 4


                          ZUPT W/2 BASES (M2)

          7.20275              7.15834          0.044411
          7.13201              7.15834        -0.0263327
          7.15324              7.15834        -0.0051064
          7.11669              7.15834        -0.0416573
          7.14982              7.15834       -0.00852089
          7.16546              7.15834        0.00711192
          7.16868              7.15834          0.010335
          7.16711              7.15834        0.00876731
          7.20367              7.15833         0.0453468
          7.15949              7.15834        0.00114539
          7.12438              7.15834        -0.0339601
          7.12775              7.15834        -0.0305962
          7.23076              7.15834         0.0724123
           7.1694              7.15834         0.0110526
          7.14557              7.15834        -0.0127723
          7.17002              7.15834         0.0116749
          7.16657              7.15834        0.00822315
          7.17329              7.15834         0.0149415
          7.11822              7.15834        -0.0401271
          7.15417              7.15834       -0.00416887
          7.12214              7.15834        -0.0361999
          7.29125              7.15834          0.132905
          7.10728              7.15834         -0.051064
          7.15111              7.15834       -0.00723693
          7.11567              7.15834        -0.0426769
          7.08974              7.15834        -0.0686072
          7.19755              7.15834         0.0392108
          7.12916              7.15836        -0.0291945
          7.19461              7.15834         0.0362638
          7.25577              7.15834          0.097428
          7.09025              7.15834        -0.0680972
          7.20021              7.15834         0.0418621
          7.16298              7.15834        0.00463132
          7.14292              7.15834        -0.0154208
          7.14574              7.15834        -0.0126082
          7.13279              7.15834        -0.0255554
          7.15601              7.15833       -0.00232236
          7.15576              7.15834       -0.00257956
          7.19948              7.15834         0.0411383
          7.15835              7.15834       2.92266e-006
          7.10394              7.15833        -0.0543925
         7.24869             7.15834       0.0903466
         7.12619             7.15834      -0.0321568
         7.08353             7.15836      -0.0748272
         7.13555             7.15834      -0.0227911
         7.21819             7.15834       0.0598437
         7.13713             7.15834      -0.0212158
         7.11653             7.15834      -0.0418092
         7.15717             7.15834     -0.00117053
         6.57505             7.15834       -0.583294

Mean Error =      -0.01235 (inches)    STD =       0.09205 (inches)
Max Error =       -0.5833 (inches)
Sampling rate = 300.03 (Hz)
Filter bandwidth = 40 (Hz)
A-to-D Range = 1 (Gs)
Integration method = 1
      (0-Simp  1-Trap  2-Adam)
Number of runs = 50
```

```
                         ZUPT W/2 BASES (M2)

            7.44339            7.15834          0.285044
            7.29352            7.15833          0.135187
            7.22559            7.15833          0.0672611
            7.17013            7.15834          0.0117896

    Mean Error =        0.1248 (inches)    STD =         0.1023 (inches)
    Max Error =         0.285 (inches)
    Sampling rate = 200 (Hz)
    Filter bandwidth = 40 (Hz)
    A-to-D Range = 1 (Gs)
    Integration method = 1
           (0-Simp  1-Trap  2-Adam)
    Number of runs = 4


                         ZUPT ONLY

            10.4659            10.7375         -0.271614
            10.4443            10.7375         -0.293167
            10.4492            10.7375         -0.288347
            10.4836            10.7375         -0.253901
            10.3753            10.7375           -0.3622
            10.5135            10.7375         -0.224022
            10.4561            10.7375          -0.28145
            10.4361            10.7375         -0.301446
            10.5301            10.7375         -0.207372
             10.488            10.7375         -0.249539
            10.4666            10.7375          -0.27089
            10.3974            10.7375         -0.340082
            10.5306            10.7375         -0.206871
            10.4628            10.7375         -0.274736
            10.4771            10.7375         -0.260416
            10.3546            10.7375         -0.382905
            10.4174            10.7375         -0.320126
            10.4964            10.7375         -0.241153
            10.5155            10.7375         -0.221975
            10.4249            10.7375         -0.312634
            10.4129            10.7375         -0.324588
            10.4964            10.7375         -0.241136
            10.5076            10.7375         -0.229934
            10.4869            10.7375         -0.250652
            10.4514            10.7375         -0.286082
            10.4267            10.7375         -0.310801
            10.4121            10.7375         -0.325385
            10.4903            10.7375         -0.247166
            10.3975            10.7375         -0.339971
            10.4454            10.7375         -0.292108
            10.5413            10.7375         -0.196196
            10.4849            10.7375          -0.25259
             10.436            10.7375         -0.301544
            10.3258            10.7375         -0.411754
            10.5613            10.7375         -0.176204
            10.4539            10.7375         -0.283599
            10.5232            10.7375         -0.214327
            10.4435            10.7375         -0.293985
            10.5209            10.7375         -0.216621
            10.4358            10.7375         -0.301699
```

```
      10.439            10.7375          -0.29853
     10.4113            10.7375         -0.326196
      10.424            10.7375         -0.313559
     10.4419            10.7375         -0.295585
      10.516            10.7375         -0.221528
     10.3946            10.7375          -0.34287
     10.4395            10.7375         -0.297976
     10.4485            10.7375         -0.289051

Mean Error =       -0.2806 (inches)    STD =       0.04909 (inches)
Max Error =       -0.4118 (inches)
Sampling rate = 200 (Hz)
Filter bandwidth = 40 (Hz)
A-to-D Range = 1 (Gs)
Integration method = 1
      (0-Simp  1-Trap  2-Adam)
Number of runs = 50
```

```
              10.6821            10.7375        -0.0553822
               10.611            10.7375         -0.126509
              10.6344            10.7375         -0.103088
              10.6443            10.7375        -0.0931978
              10.6966            10.7375        -0.0409445
              10.6544            10.7375        -0.0831523
              10.7071            10.7375        -0.0303822
               10.644            10.7375        -0.0935399
              10.6869            10.7375        -0.0506013

Mean Error =       -0.3825 (inches)    STD =          1.617 (inches)
Max Error =       -9.25535 (inches)
Sampling rate = 200 (Hz)
Filter bandwidth = 40 (Hz)
A-to-D Range = 1 (Gs)
Integration method = 1
      (0-Simp  1-Trap  2-Adam)
Number of runs = 50
```

```
                           ZUPT W/2 BASES (M2)

       7.43025             7.15834           0.271904
       7.39292             7.15834           0.234581
       7.21671             7.15834           0.0583694
       7.20442             7.15834           0.0460733

Mean Error =        0.1527 (inches)     STD =        0.1015 (inches)
Max Error =        0.2719 (inches)
Sampling rate = 200 (Hz)
Filter bandwidth = 40 (Hz)
A-to-D Range = 1 (Gs)
Integration method = 1
      (0-Simp  1-Trap  2-Adam)
Number of runs = 4


                           SINGLE

        3.7217             3.57917           0.142527
       3.74455             3.57917           0.165374
       3.67074             3.57917           0.0915699
       3.76825             3.57917           0.189078
       3.71378             3.57917           0.134606
       3.79898             3.57917           0.219806
       3.76927             3.57916           0.190115
        3.7909             3.57917           0.211731
       3.81156             3.57917           0.232383
       3.77446             3.57917           0.195283
       3.84755             3.57917           0.268377
       3.68388             3.57917           0.104713
       3.83674             3.57917           0.257572
       3.75973             3.57917           0.180559
       3.84719             3.57917           0.268018
       3.71489             3.57917           0.135715
       3.77763             3.57917           0.198457
       3.71851             3.57917           0.139334
        3.6254             3.57916           0.0462468
       3.81362             3.57917           0.234453
       3.74733             3.57917           0.168155
        3.7395             3.57917           0.160329
       3.70978             3.57917            0.13061
       3.68336             3.57917           0.104189
       3.75655             3.57917           0.177379
       3.71036             3.57916           0.131202
       3.67549             3.57916           0.0963378
       3.63095             3.57917           0.0517735
       3.68743             3.57917           0.108262
       3.72398             3.57916           0.144821
       3.70681             3.57916           0.127657
       3.74794             3.57917           0.168766
       3.71601             3.57919           0.136825
        3.6639             3.57917           0.0847326
       3.76259             3.57917           0.183421
       3.68004             3.57916           0.100881
       3.70031             3.57917           0.121137
       3.72125             3.57916           0.142093
       3.71998             3.57916            0.14082
       3.66858             3.57917           0.0894032
```

```
           3.60977             3.57919         0.0305816
           3.67302             3.57916         0.0938641
           3.77422             3.57917          0.195046
           3.72409             3.57917          0.144922
           3.70178             3.57916          0.122626
           3.76619             3.57917          0.187021
            3.7526             3.57917           0.17343
           3.75786             3.57917          0.178684
           3.65471             3.57917         0.0755369

    Mean Error =       0.1497 (inches)    STD =       0.05474 (inches)
    Max Error =       0.2684 (inches)
    Sampling rate = 200 (Hz)
    Filter bandwidth = 40 (Hz)
    A-to-D Range = 1 (Gs)
    Integration method = 1
          (0-Simp  1-Trap  2-Adam)
    Number of runs = 50
```

```
                         ZUPT W/2 BASES (M2)

        7.71103              7.15834           0.552688
        7.40671              7.15834           0.248369

Mean Error =        0.4005 (inches)    STD =        0.1522 (inches)
Max Error =       0.5527 (inches)
Sampling rate = 200 (Hz)
Filter bandwidth = 40 (Hz)
A-to-D Range = 1 (Gs)
Integration method = 1
     (0-Simp  1-Trap  2-Adam)
Number of runs = 2
```

```
                         ZUPT W/2 BASES (M2)

        7.33207              7.15834           0.173723
        7.20969              7.15834          0.0513454
        7.19432              7.15834          0.0359726
        7.20165              7.15834          0.0433022
         7.0976              7.15834         -0.0607487
        7.18628              7.15834           0.027941
        7.12231              7.15834         -0.0360384
        7.14922              7.15834        -0.00912419
         7.2457              7.15834          0.0873563
        7.09991              7.15834         -0.0584323
        7.13646              7.15834         -0.0218851
        7.36029              7.15834            0.20195
        7.08902              7.15833         -0.0693051
        7.08859              7.15836         -0.0697731
        7.08235              7.15834         -0.0759961
        8.48598              7.15834            1.32764
        6.61151              7.15834          -0.546837
        6.73718              7.15834          -0.421167
        7.07975              7.15836         -0.0786095
        7.14799              7.15834          -0.010349
        7.21924              7.15834          0.0608971
         7.0986              7.15834          -0.059746
        7.08265              7.15834         -0.0756949
        7.12416              7.15834          -0.034186
        7.24217              7.15834          0.0838257
        7.22167              7.15834           0.063328
        7.10807              7.15834         -0.0502786
        7.14702              7.15834         -0.0113227
        7.17431              7.15834           0.015971
        7.10684              7.15834          -0.051503
        7.15627              7.15834        -0.00207368
        7.13143              7.15834         -0.0269168
        7.13841              7.15834         -0.0199348
        7.18337              7.15834          0.0250295
        7.15283              7.15834        -0.00551456
        7.17664              7.15834          0.0182948
        7.20265              7.15834          0.0443014
        7.11525              7.15834          -0.043093
        7.15147              7.15834        -0.00687234
        7.17072              7.15834          0.0123791
        7.17711              7.15834          0.0187615
        7.19479              7.15834          0.0364444
        7.11258              7.15834         -0.0457646
```

```
      7.13856           7.15834        -0.0197797
      7.12869           7.15834        -0.0296585
      7.19144           7.15834         0.0330956
      7.13001           7.15834        -0.0283296
      7.16615           7.15833        0.00781981
      7.15237           7.15834         -0.005978

Mean Error =     0.007973 (inches)    STD =        0.2189 (inches)
Max Error =        1.328 (inches)
Sampling rate = 200 (Hz)
Filter bandwidth = 40 (Hz)
A-to-D Range = 1 (Gs)
Integration method = 1
      (0-Simp  1-Trap  2-Adam)
Number of runs = 50
```

```
                          ZUPT W/2 BASES (M2)

          7.45049           7.15834          0.292143
          7.28482           7.15834          0.126476
          7.21856           7.15834         0.0602202
          7.15354           7.15834       -0.00480726

Mean Error =       0.1185 (inches)    STD =       0.1105 (inches)
Max Error =      0.2921 (inches)
Sampling rate = 300.03 (Hz)
Filter bandwidth = 40 (Hz)
A-to-D Range = 1 (Gs)
Integration method = 1
      (0-Simp  1-Trap  2-Adam)
Number of runs = 4


                          SINGLE

          3.55978           3.57917          -0.01939
          3.54229           3.57917        -0.0368816
          3.54694           3.57917        -0.0322323
          3.63482           3.57917         0.0556449
          3.63281           3.57917         0.0536336
          3.62732           3.57916         0.0481628
          3.65493           3.57917         0.0757594
          3.64363           3.57917         0.0644545
          3.58617           3.57917        0.00700136
          3.65271           3.57917         0.0735389
          3.64166           3.57917         0.0624923
          3.57481           3.57916       -0.00434921
          3.63181           3.57917         0.0526363
           3.6761           3.57917         0.0969242
          3.69462           3.57916           0.11546
          3.63971           3.57917         0.0605346
          3.51247           3.57917        -0.0667012
          3.63727           3.57917         0.0580935
          3.60522           3.57917         0.0260435
          3.54944           3.57917        -0.0297321
          3.54651           3.57917        -0.0326585
          3.60893           3.57917         0.0297585
          3.64881           3.57917         0.0696343
          3.63893           3.57917         0.0597571
          3.62365           3.57917          0.044475
          3.63384           3.57917          0.054666
          3.67087           3.57917         0.0916981
          3.59821           3.57917         0.0190411
          3.68936           3.57917           0.11019
          3.70367           3.57917          0.124498
          3.69755           3.57917          0.118376
          3.68552           3.57917          0.106345
          3.65328           3.57917         0.0741094
          3.65864           3.57916         0.0794848
          3.73494           3.57917          0.155771
          3.71158           3.57916          0.132425
          3.61414           3.57917         0.0349671
          3.67858           3.57917         0.0994099
          3.64598           3.57917         0.0668099
          3.69938           3.57917          0.120206
          3.68181           3.57916          0.102653
```

```
              3.76603           3.57916          0.18687
              3.72948           3.57917          0.150306
              3.69942           3.57916          0.120266
              3.57036           3.57919         -0.00883179
              3.69021           3.57917          0.111038
              3.61883           3.57917          0.0396579
              3.64885           3.57917          0.069681
              3.61286           3.57917          0.0336911
              3.61364           3.57917          0.0344713

       Mean Error =         0.0612 (inches)     STD =        0.05404 (inches)
       Max Error =         0.1869 (inches)
       Sampling rate = 300.03 (Hz)
       Filter bandwidth = 40 (Hz)
       A-to-D Range = 1 (Gs)
       Integration method = 1
             (0-Simp  1-Trap  2-Adam)
       Number of runs = 50
```